Carnegie Mellon Univ.  
Dept. of Computer Science  
15-415/615 – DB Applications  

C. Faloutsos  
Lecture#5: Relational calculus

General Overview - rel. model

• history  
• concepts  
• Formal query languages  
  – relational algebra  
  – rel. tuple calculus  
  – rel. domain calculus

Overview - detailed

• rel. tuple calculus  
  – why?  
  – details  
  – examples  
  – equivalence with rel. algebra  
  – more examples; ‘safety’ of expressions  
• rel. domain calculus + QBE
Motivation

- Q: weakness of rel. algebra?
- A: procedural
  - describes the steps (i.e., "how")
  - (still useful, for query optimization)

Solution: rel. calculus

- describes what we want
- two equivalent flavors: 'tuple' and 'domain'
- basis for SQL and QBE, resp.
- Useful for proofs (see query optimization, later)

Rel. tuple calculus (RTC)

- first order logic

\[ \{ t \mid P(t) \} \]

'Give me tuples 't', satisfying predicate P - eg:

\[ \{ t \mid t \in \text{STUDENT} \} \]
Details

- symbols allowed:
  \( \land, \lor, \neg, \Rightarrow, \rightarrow, \leftarrow, =, \neq, =, \neq, (, ), \in \)

- quantifiers: \( \forall, \exists \)

Specifically

- Atom
  
  \( t \in \text{TABLE} \)
  
  \( t.\text{attr} \leq \text{const} \)
  
  \( t.\text{attr} \leq s.\text{attr} \)

Specifically

- Formula:
  - atom
  - if \( P_1, P_2 \) are formulas, so are \( P_1 \land P_2, P_1 \lor P_2 \ldots \)
  - if \( P(s) \) is a formula, so are \( \exists s(P(s)) \)
  - \( \forall s(P(s)) \)
Specifically

- Reminders:
  - DeMorgan: \( P_1 \land P_2 = \neg (\neg P_1 \lor \neg P_2) \)
  - implication: \( P_1 \Rightarrow P_2 = \neg P_1 \lor P_2 \)
  - double negation:
    \[ \forall s \in \text{TABLE} \ (P(s)) = \neg \exists s \in \text{TABLE} \ (\neg P(s)) \]
    ‘every human is mortal : no human is immortal’

Reminder: our Mini-U db

<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 all student records

\[ \{ t \mid t \in \text{STUDENT} \} \]

output tuple of type ‘STUDENT’
Examples

• (selection) find student record with ssn=123

\{t \mid t \in \text{STUDENT} \land t.ssn = 123\}

• (projection) find name of student with ssn=123

\{t \mid t \in \text{STUDENT} \land t.ssn = 123\}
Examples

• (projection) find name of student with ssn=123

\[
\{ t \mid \exists s \in \text{STUDENT} (s.ssn = 123 \land t.name = s.name) \}
\]

't' has only one column

‘Tracing’

\[
\{ t \mid \exists s \in \text{STUDENT} (s.ssn = 123 \land t.name = s.name) \}
\]

Examples cont’d

• (union) get records of both PT and FT students
Examples cont’d

• (union) get records of both PT and FT students

\[ \{ t \mid t \in FT\_STUDENT \lor t \in PT\_STUDENT \} \]

Examples

• difference: find students that are not staff

(assuming that STUDENT and STAFF are union-compatible)

Examples

• difference: find students that are not staff

\[ \{ t \mid t \in STUDENT \land t \notin STAFF \} \]
Cartesian product

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

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

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

Cartesian product

- find all the pairs of (male, female)

\[
\{ t \mid \exists m \in \text{MALE} \land \\
\exists f \in \text{FEMALE} \\
\text{t.m.name} = m.\text{name} \land \\
\text{t.f.name} = f.\text{name}\}
\]

‘Proof’ of equivalence

- rel. algebra <-> rel. tuple calculus
Overview - detailed

- rel. tuple calculus
  - why?
  - details
  - examples
  - equivalence with rel. algebra
  - more examples; ‘safety’ of expressions
- re. domain calculus + QBE

More examples

- join: find names of students taking 15-415

Reminder: our Mini-U db

<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>Jones</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>
More examples

• join: find names of students taking 15-415

\[
\{ t \mid \exists s \in \text{STUDENT} \\
\quad \land \exists e \in \text{TAKES} \ (s, ssn = e, ssn \land \\
\quad t.name = s.name \land \\
\quad e.c - id = 15 - 415) \}
\]

More examples

• join: find names of students taking 15-415

\[
\{ t \mid \exists s \in \text{STUDENT} \\
\quad \land \exists e \in \text{TAKES} \ (s, ssn = e, ssn \land \\
\quad t.name = s.name \land \\
\quad e.c - id = 15 - 415) \}
\]

More examples

• 3-way join: find names of students taking a 2-unit course
Reminder: our Mini-U db

<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>main str</td>
<td>2</td>
</tr>
<tr>
<td>234</td>
<td>15-412</td>
</tr>
<tr>
<td>jones</td>
<td>o.s.</td>
</tr>
<tr>
<td>forbes ave</td>
<td>2</td>
</tr>
</tbody>
</table>

More examples

- 3-way join: find names of students taking a 2-unit course

\[
\begin{align*}
&\{t\mid \exists s \in STUDENT \land \exists e \in TAKES \\
&\exists c \in CLASS (s.csn = e.csn \land \\
&e.c = c.c \land e.s = t.s) \land \\
&t.name = s.name \land \\
&c.units = 2 \}\end{align*}
\]


More examples

- 3-way join: find names of students taking a 2-unit course - in rel. algebra??

\[
\pi_{name}(\sigma_{units=2}(STUDENT \bowtie_{\bowrightarrow} TAKES \bowtie_{\bowrightarrow} CLASS))
\]
Even more examples:

- self-joins: find Tom’s grandparent(s)

<table>
<thead>
<tr>
<th>p-id</th>
<th>c-id</th>
</tr>
</thead>
<tbody>
<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>

Hard examples: DIVISION

- find suppliers that shipped all the ABOMB parts

<table>
<thead>
<tr>
<th>SHIPMENT</th>
<th>ABOMB</th>
<th>BAD_S</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td>p3</td>
<td></td>
</tr>
</tbody>
</table>

\[ \{ t \mid \exists p \in PC \land \exists q \in PC \\
\{ \ p . c - id = q . p - id \land \\
\ p . p - id = t . p - id \land \\
\ q . c - id = \text{"Tom"} \} \} \]
Hard examples: DIVISION

- find suppliers that shipped all the ABOMB parts

\[ \{ t \mid \forall p(p \in ABOMB \Rightarrow ( \exists s \in SHIPMENT ( t.s# = s.s# \land s.p# = p.p# ))) \} \]

General pattern

- three equivalent versions:
  - 1) if it’s bad, he shipped it
    \[ \{ t \mid \forall p(p \in ABOMB \Rightarrow (P(t)) \} \]
  - 2) either it was good, or he shipped it
    \[ \{ t \mid \forall p(p \notin ABOMB \lor (P(t)) \} \]
  - 3) there is no bad shipment that he missed
    \[ \{ t \mid \neg \exists p(p \in ABOMB \land (\neg P(t)) \} \]

\[ a \Rightarrow b \] is the same as \( \neg a \lor b \)

- If \( a \) is true, \( b \) must be true for the implication to be true. If \( a \) is true and \( b \) is false, the implication evaluates to false.
- If \( a \) is not true, we don’t care about \( b \), the expression is always true.
More on division

- find (SSNs of) students that take all the courses that ssn=123 does (and maybe even more)
  find students ‘s’ so that
  if 123 takes a course => so does ‘s’

Safety of expressions

- FORBIDDEN: \( \{t \mid t \notin \text{STUDENT} \} \)
  It has infinite output!!
- Instead, always use
  \( \{t \mid t \in \text{SOME \text{\{-TABLE\}}} \} \)
Overview - conclusions

- rel. tuple calculus: DECLARATIVE
  - dfn
  - details
  - equivalence to rel. algebra
- rel. domain calculus + QBE

General Overview

- relational model
- Formal query languages
  - relational algebra
  - rel. tuple calculus
  - rel. domain calculus

Rel. domain calculus (RDC)

- Q: why?
- A: slightly easier than RTC, although equivalent - basis for QBE.
- idea: domain variables (w/ F.O.L.) - eg:
- ‘find STUDENT record with ssn=123’
Rel. Dom. Calculus

• find STUDENT record with ssn=123

\{ <s,n,a> | <s,n,a> \in STUDENT \land s = 123 \}

Details

• Like R.T.C - symbols allowed:
  \( \land, \lor, \neg, \Rightarrow, \rightarrow, \leftarrow, \leftrightarrow, \in, \),

• quantifiers \( \forall, \exists \)

Details

• but: domain (= column) variables, as opposed to tuple variables, eg:

\[ <s,n,a> \in STUDENT \]

\begin{itemize}
  \item ssn
  \item name
  \item address
\end{itemize}
Reminder: our Mini-U db

<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

• (selection) find student record with ssn=123

\{ <123, n, a> | \langle s, n, a \rangle \in STUDENT \}

or

\{ <s, n, a> | \langle s, n, a \rangle \in STUDENT \land s = 123 \}

RTC: \{ t | t \in STUDENT \land t.ssn = 123 \}

Examples

• (projection) find name of student with ssn=123

\{ <n> | \langle 123, n, a \rangle \in STUDENT \}

need to ‘restrict’ “a”

Examples

• (projection) find name of student with ssn=123

\{ <n> | \exists a(\langle 123, n, a \rangle \in STUDENT) \}

RTC: \{ t | \exists s \in STUDENT (s.ssn = 123 \land t.name = s.name) \}
Examples cont’d

• (union) get records of both PT and FT students

RTC: \( \{ t | t \in \text{FT\_STUDENT} \lor t \in \text{PT\_STUDENT} \} \)

Examples cont’d

• (union) get records of both PT and FT students

\( \{ <s,n,a> | <s,n,a> \in \text{FT\_STUDENT} \lor <s,n,a> \in \text{PT\_STUDENT} \} \)

Examples

• difference: find students that are not staff

RTC: \( \{ t | t \in \text{STUDENT} \land t \notin \text{STAFF} \} \)
Examples

- difference: find students that are not staff

\{< s, n, a >\mid < s, n, a >\in\text{STUDENT} \land < s, n, a >\notin\text{STAFF}\}

Cartesian product

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

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

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

Cartesian product

- find all the pairs of (male, female) - RTC:

\[
\{t \mid \exists m \in \text{MALE} \land \\
\exists f \in \text{FEMALE} \\
\exists t.m = m.name \land \\
\exists t.f = f.name\}
\]
Cartesian product

• find all the pairs of (male, female) - RDC:

\[ \{ < m, f > | < m > \in \text{MALE} \land < f > \in \text{FEMALE} \} \]

‘Proof’ of equivalence

• rel. algebra <-> rel. domain calculus
  <-> rel. tuple calculus

Overview - detailed

• rel. domain calculus
  – why?
  – details
  – examples
  – equivalence with rel. algebra
  – more examples; ‘safety’ of expressions
More examples

- join: find names of students taking 15-415

Reminder: our Mini-U db

<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>

More examples

- join: find names of students taking 15-415 - in RTC

\{ s | \exists e \in \text{TAKES} ( s.ssn = e.ssn \wedge t.name = s.name \wedge e.c = id = 15 - 415) \}
More examples

• join: find names of students taking 15-415 in RDC

\{ < n > | \exists a \exists g ( < s, n, a > \in \text{STUDENT} \\
\land < s, 15 - 415, g > \in \text{TAKES} ) \}

Sneak preview of QBE:

\{ < n > | \exists a \exists g ( < s, n, a > \in \text{STUDENT} \\
\land < s, 15 - 415, g > \in \text{TAKES} ) \}

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>x</td>
<td>P</td>
</tr>
</tbody>
</table>

Sneak preview of QBE:

• very user friendly
• heavily based on RDC
• very similar to MS Access interface

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>x</td>
<td>P</td>
</tr>
</tbody>
</table>
More examples

- 3-way join: find names of students taking a 2-unit course - in RTC:

\[ \{ t \} \exists s \in \text{STUDENT} \land \exists e \in \text{TAKES} \]
\[ \exists c \in \text{CLASS} \ (s.ssn = e.ssn \land e.c = c.c \land t.name = s.name \land c.units = 2) \]

Reminder: our Mini-U db

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>c-id</td>
</tr>
<tr>
<td>Smith</td>
<td>15-413</td>
</tr>
<tr>
<td>Jones</td>
<td>15-412</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>B</td>
</tr>
</tbody>
</table>

More examples

- 3-way join: find names of students taking a 2-unit course

\[ \{ \langle n \rangle \} \]
\[ \langle s, n, a > \in \text{STUDENT} \land \langle s, c, g > \in \text{TAKES} \land \langle c, cn, 2 > \in \text{CLASS} \} \]
More examples

• 3-way join: find names of students taking a 2-unit course

\[
\{ n \mid \exists s, a, c, g, cn \\
\quad \quad \quad \quad n \in \text{STUDENT} \land \\
\quad \quad \quad \quad s \in \text{TAKES} \land \\
\quad \quad \quad \quad c, g, 2 \in \text{CLASS} \}
\]

Even more examples:

• self-joins: find Tom’s grandparent(s)

<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>

Even more examples:

• self-joins: find Tom’s grandparent(s)

\[
\{ t \mid \exists p \in \text{PC} \land \exists q \in \text{PC} \\
\quad \quad \quad \quad p.c - id = q.p - id \land \\
\quad \quad \quad \quad p.p - id = t.p - id \land \\
\quad \quad \quad \quad q.c - id = "Tom" \}
\]
Even more examples:

- **self-joins**: find Tom's grandparent(s)

\[
\{ \langle g \rangle \mid \exists p \in PC \land \exists q \in PC \\
( p \cdot e = q \cdot e \land \\
q \cdot e = t \cdot e \land \\
p \cdot e = \langle T, \text{Tom} \rangle) \}
\]

Even more examples:

- **self-joins**: find Tom's grandparent(s)

\[
\{ \langle g \rangle \mid \exists p \in PC \land \\
\langle p, \text{Tom} \rangle \in PC \}
\]

**Hard examples: DIVISION**

- find suppliers that shipped all the ABOMB parts

<table>
<thead>
<tr>
<th>SHIPMENT</th>
<th>ABOMB</th>
<th>BAD_S</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td>p3</td>
<td></td>
</tr>
</tbody>
</table>

+ ABOMB

<table>
<thead>
<tr>
<th>p1</th>
<th>p1</th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= BAD_S

| s1 | | | | |
|----| | | | |

| s1 | | | | |
|----| | | | |
Hard examples: DIVISION

• find suppliers that shipped all the ABOMB parts

\[
\{ t \mid \forall p ( p \in ABOMB \implies ( \exists s \in SHIPMENT ( t.s# = s.s# \land s.p# = p.p# )) ) \}
\]
More on division

- find students that take all the courses that ssn=123 does (and maybe even more)

\{<s> | \forall c (\exists g (<s,c,g> \in \text{TAKES}) \Rightarrow \\
\exists g'(<s,c,g'>) \in \text{TAKES})} \}

Safety of expressions

- similar to RTC
- FORBIDDEN:

\{<s,n,a> |<s,n,a> \notin \text{STUDENT} \}

Overview - detailed

- rel. domain calculus + QBE
  - dfn
  - details
  - equivalence to rel. algebra
Fun Drill: Your turn …

- Schema:
  - Movie(title, year, studioName)
  - ActsIn(movieTitle, starName)
  - Star(name, gender, birthdate, salary)

Your turn …

- Queries to write in TRC:
  - Find all movies by Paramount studio
  - … movies starring Kevin Bacon
  - Find stars who have been in a film w/Kevin Bacon
  - Stars within six degrees of Kevin Bacon*
  - Stars connected to K. Bacon via any number of films**

* Try two degrees for starters  ** Good luck with this one!

Answers …

- Find all movies by Paramount studio

\{M | M∈Movie ∧ M.studioName = ‘Paramount’\}
Answers …

• Movies starring Kevin Bacon

\{M | M \in \text{Movie} \land \\
\exists A \in \text{ActsIn}(A.\text{movieTitle} = M.\text{title} \land \\
A.\text{starName} = 'Bacon')\}\}

Answers …

• Stars who have been in a film w/Kevin Bacon

\{S | S \in \text{Star} \land \\
\exists A \in \text{ActsIn}(A.\text{starName} = S.\text{name} \land \\
\exists A2 \in \text{ActsIn}(A2.\text{movieTitle} = A.\text{movieTitle} \land \\
A2.\text{starName} = 'Bacon')\}\}

Answers …

• Stars within six degrees of Kevin Bacon

\{S | S \in \text{Star} \land \\
\exists A \in \text{ActsIn}(A.\text{starName} = S.\text{name} \land \\
\exists A2 \in \text{ActsIn}(A2.\text{movieTitle} = A.\text{movieTitle} \land \\
A2.\text{starName} = 'Bacon') \land \\
\exists A3 \in \text{ActsIn}(A3.\text{starName} = A2.\text{starName} \land \\
\exists A4 \in \text{ActsIn}(A4.\text{movieTitle} = A3.\text{movieTitle} \land \\
A4.\text{starName} = 'Bacon')\}\}

\text{two}
Two degrees:

S: name

A3: movie star
A4: movie star

“Bacon”

Answers ...

- Stars connected to K. Bacon via any number of films
- Sorry … that was a trick question
  - Not expressible in relational calculus!!
- What about in relational algebra?
  - No – RA, RTC, RDC are equivalent
Expressive Power

- Expressive Power (Theorem due to Codd):
  - Every query that can be expressed in relational algebra can be expressed as a safe query in DRC/TRC; the converse is also true.

- Relational Completeness:
  Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus. (actually, SQL is more powerful, as we will see…)

Summary

- The relational model has rigorously defined query languages — simple and powerful.
- Relational algebra is more operational/procedural
  - useful as internal representation for query evaluation plans
- Relational calculus is declarative
  - users define queries in terms of what they want, not in terms of how to compute it.

Summary - cnt’d

- Several ways of expressing a given query
  - a query optimizer should choose the most efficient version.
- Algebra and safe calculus have same expressive power
  - leads to the notion of relational completeness.