Database Applications (15-415)

Relational Calculus
Lecture 6, January 27, 2015

Mohammad Hammoud
Today...

- **Last Session:**
  - Relational Algebra

- **Today’s Session:**
  - Relational calculus
    - Relational tuple calculus

- **Announcements:**
  - PS2 is now posted. Due on Feb 08, 2015 by midnight
  - PS1 grades are out
  - In the next recitation we will practice on relational algebra and calculus
Overview - Detailed

- Relational Tuple Calculus (RTC)
  - Why?
  - Details
  - Examples
  - Equivalence with relational algebra
  - ‘Safety’ of expressions
Motivation

- **Question**: What is the main “weakness” of relational algebra?

- **Answer**: Procedural
  - It describes the steps (i.e., ‘how’)
  - Still useful, especially for query optimization
Relational Calculus (in General)

- It describes what we want (not how)
- It has two equivalent flavors, ‘tuple’ and ‘domain’ calculus
  - We will only focus on relational ‘tuple’ calculus
- It is the basis for SQL and Query By Example (QBE)
- It is useful for proofs (see query optimization, later)
Relational Tuple Calculus (RTC)

- RTC is a subset of ‘first order logic’:

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

A “formula” that describes \( t \)

Give me tuples ‘t’, satisfying predicate ‘P’

- Examples:
  - Find all students: \( \{ t \mid t \in STUDENT \} \)
  - Find all sailors with a rating above 7:
    \( \{ t \mid t \in Sailors \land t.rating > 7 \} \)
Syntax of RTC Queries

- The allowed symbols:

\[ \land, \lor, \neg, \Rightarrow, \geq, \leq, \neq, =, \neg, \land, \lor, (, ), \in \]

- Quantifiers:

\[ \forall, \exists \]
Syntax of RTC Queries

- Atomic “formulas”:

  \[ t \in \text{TABLE} \]
  \[ t.\text{attr} \ op \ const \]
  \[ t.\text{attr} \ op \ s.\text{attr} \]

Where \( \text{op} \) is an operator in the set \{<, >, =, \leq, \geq, \neq \}
Syntax of RTC Queries

- A “formula” is:
  - Any atomic formula

- If $P_1$ and $P_2$ are formulas, so are

  \[ \neg P_1; \ \neg P_2; \ P_1 \land P_2; \ P_1 \lor P_2; \ P_1 \Rightarrow P_2 \]

- If $P(s)$ is a formula, so are

  \[ \exists s(P(s)) \]
  \[ \forall s(P(s)) \]
Basic Rules

- Reminders:
  - De Morgan: \( P_1 \land P_2 \equiv \neg(\neg P_1 \lor \neg P_2) \)
  - Implication: \( P_1 \Rightarrow P_2 \equiv \neg P_1 \lor P_2 \)
  - Double Negation:

\[
\forall s \in TABLE \ (P(s)) \equiv \neg \exists s \in TABLE \ (\neg P(s))
\]

'every human is mortal: no human is immortal'
A Mini University Database

<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>2</td>
</tr>
<tr>
<td>forbes ave</td>
<td>2</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

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

output
tuple of type ‘STUDENT’
Examples

- Find the student record with ssn=123
Examples

- Find the student record with ssn=123

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

This is equivalent to the ‘Selection’ operator in Relational Algebra!
Examples

- Find the **name** of the student with ssn=123

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

**Will this work?**
Examples

- Find the **name** of the student with ssn=123

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

‘t’ has only one column

This is equivalent to the ‘Projection’ operator in Relational Algebra!
Examples

- Get records of both part time and full time students:

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

This is equivalent to the ‘Union’ operator in Relational Algebra!

* Assume we maintain tables for PT\_STUDENT and FT\_STUDENT in our Mini University DB
Examples

- Find students that are not staff*

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

This is equivalent to the ‘Difference’ operator in Relational Algebra!

* Assume we maintain a table for STAFF in our Mini University DB and that STUDENT and STAFF are union-compatible
Cartesian Product: A Reminder

- Assume MALE and FEMALE dog tables as follows:

<table>
<thead>
<tr>
<th>MALE</th>
<th>FEMALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>spike</td>
<td>lassie</td>
</tr>
<tr>
<td>spot</td>
<td>shiba</td>
</tr>
</tbody>
</table>

\[ \text{MALE} \times \text{FEMALE} \]

This gives all possible couples!
Examples (Cont’d)

- Find all the pairs of (male, female) dogs

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

This is equivalent to the ‘Cartesian Product’ operator in Relational Algebra!
More Examples

- Find the names of students taking 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>2</td>
</tr>
<tr>
<td>forbes ave</td>
<td>2</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>

2-way Join!
More Examples

• Find the names of students taking 15-415

\[ \{ t \mid \exists s \in STUDENT \wedge \exists e \in TAKES \ ( s.ssn = e.ssn \wedge t.name = s.name \wedge e.c - id = 15 - 415) \} \]
More Examples

• Find the names of students taking 15-415

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

- **Find the names of students taking a 2-unit course**

### 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>

### CLASS

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</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>

---

3-way Join!
More Examples

• Find the names of students taking a 2-unit course

\{ t | \exists s \in STUDENT \land \exists e \in TAKES \\
\exists c \in CLASS ( s.ssn = e.ssn \land \\
e.c - id = c.c - id \land \\
t.name = s.name \land \\
c.units = 2 ) \}\n
What is the equivalence of this in Relational Algebra?

More on Joins

- Assume a Parent-Children (PC) table instance as follows:

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

- Who are Tom’s grandparent(s)? (*this is a self-join*)
More Join Examples

- Find Tom’s grandparent(s)

\[
\{ t \mid \exists p \in PC \land \exists q \in PC \\
( p.c - id = q.p - id \land \\
p.p - id = t.p - id \land \\
q.c - id = "Tom" ) \}
\]

What is the equivalence of this in Relational Algebra?
Harder Examples: DIVISION

- Find suppliers that shipped all the bad parts

<table>
<thead>
<tr>
<th>SHIPMENT</th>
<th>BAD_P</th>
<th>BAD_S</th>
</tr>
</thead>
<tbody>
<tr>
<td>s#</td>
<td>p#</td>
<td></td>
</tr>
<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>

\[ s1 = \frac{\text{BAD_P}}{\text{BAD_S}} \]
Harder Examples: DIVISION

- Find suppliers that shipped all the bad parts

\[ \{ t \mid \forall p (p \in BAD \_ P \Rightarrow (\exists s \in SHIPMENT( t.s# = s.s# \land s.p# = p.p#)))) } \]
General Patterns

- There are three equivalent versions:
  1) If it is bad, he shipped it

\[ \{ t \mid \forall p (p \in BAD \_ P \Rightarrow (P(t))) \} \]

  2) Either it was good, or he shipped it

\[ \{ t \mid \forall p (p \notin BAD \_ P \lor (P(t))) \} \]

  3) There is no bad shipment that he missed

\[ \{ t \mid \neg \exists p (p \in BAD \_ P \land (\neg P(t))) \} \]
More on Division

- Find (SSNs of) students who are taking all the courses that ssn=123 is (and maybe even more)

One way to think about this:
Find students ‘s’ so that if 123 takes a course => so does ‘s’
More on Division

- Find (SSNs of) students who are taking all the courses that ssn=123 is (and maybe even more)

\[
\{ o \mid \forall t((t \in TAKES \land t.ssn = 123) \Rightarrow \\
\exists t1 \in TAKES (\\
\quad t1.c - id = t.c - id \land \\
\quad t1.ssn = o.ssn) \\
) \}
\]
‘Proof’ of Equivalence

- Relational Algebra <-> RTC

But...
Safety of Expressions

- FORBIDDEN:

\[ \{ t \mid t \not\in \text{STUDENT} \} \]

It has infinite output!!

- Instead, always use:

\[ \{ t \mid \ldots t \in \text{SOME} - \text{TABLE} \} \]
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

- 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
Next Class

SQL- Part I