Database Applications (15-415)

Relational Calculus
Lecture 5, January 27, 2014

Mohammad Hammoud
Today...

- **Last Session:**
  - Relational Algebra

- **Today’s Session:**
  - Relational algebra
    - The division operator and summary
  - Relational calculus
    - Tuple relational calculus
    - Domain relational calculus

- **Announcement:**
  - PS2 will be posted by tonight. It is due on Feb 06, 2014 by midnight
Outline

The Division Operator and Summary of Relational Operators

Tuple Relational Calculus

Domain Relational Calculus
The Division Operation

- **Division**: $R \div S$
  - Not supported as a primitive operator, but useful for expressing queries like:
    - Find sailors who have reserved all boats
  - Let $A$ have 2 fields, $x$ and $y$; $B$ has only field $y$:
    - $A/B$ contains all $x$ tuples (sailors) such that for every $y$ tuple (boat) in $B$, there is an $xy$ tuple in $A$
    - Or: If the set of $y$ values (boats) associated with an $x$ value (sailor) in $A$ contains all $y$ values in $B$, then $x$ value is in $A/B$
  - Formally: $A/B = \{ \langle x \rangle | \exists \langle x, y \rangle \in A \ \land \ \forall \langle y \rangle \in B \}$
  - In general, $x$ and $y$ can be any lists of fields; $y$ is the list of fields in $B$, and $x$ $y$ is the list of fields in $A$
Examples of Divisions

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>

A

B1

- pno
  - p1
  - p2
  - p3
  - p4

B2

- pno
  - p1
  - p2
  - p4

B3

- pno
  - p1
  - p2
  - p4

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
</tbody>
</table>

A/B1

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s1</td>
</tr>
</tbody>
</table>

A/B2

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s1</td>
</tr>
</tbody>
</table>

A/B3
Expressing A/B Using Basic Operators

- Division can be derived from the fundamental operators

- Idea: For A/B, compute all x values that are not `disqualified’ by some y value in B
  - x value is disqualified if by attaching y value from B, we obtain an xy tuple that is “not” in A

Disqualified x values: \( \pi_x (\pi_x(A) \times B) - A \)

A/B: \( \pi_x (A) - \) all disqualified tuples
A Query Example

- Find the names of sailors who’ve reserved all boats

\[
\rho (\text{Tempsids}, (\pi_{\text{sid}, \text{bid}} \text{Reserves}) / (\pi_{\text{bid}} \text{Boats}))
\]

\[
\pi_{\text{sname}} (\text{Tempsids} \bowtie \text{Sailors})
\]

How can we find sailors who’ve reserved all ‘Interlake’ boats?
Relational Algebra: Summary

Operators (with notations):

1. **Selection** ($\sigma$): selects a subset of rows from a relation

2. **Projection** ($\pi$): deletes unwanted columns from a relation

3. **Cross-product** ($\times$): allows combining two relations

4. **Set-difference** (−): retains tuples which are in relation 1, “but not” in relation 2

5. **Union** ($\cup$): retains tuples which are in “either” relation 1 “or” relation 2, “or in both”
Relational Algebra: Summary

- **Operators (with notations):**
  6. **Intersection** (\( \cap \)): retains tuples which are in relation 1 “and” in relation 2
  7. **Join** (\( \Join \)): allows combining two relations according to a specific condition (e.g., *theta*, *equi* and *natural* joins)
  8. **Division** (\( \div \)): generates the largest instance Q such that Q \( \times \) B \( \subseteq \) A when computing A/B
  9. **Renaming** (\( \rho \)): returns an instance of a new relation with some fields being potentially “renamed”
Outline

- The Division Operator and Summary of Relational Operators
- Tuple Relational Calculus
- Domain Relational Calculus
Overview - Detailed

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

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

- **Answer**: Procedural
  - It describes the steps for computing the desired answer (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

- It is the basis for SQL and Query By Example (QBE)

- It is useful for proofs (see query optimization, later)
Tuple Relational Calculus (TRC)

- RTC is a subset of ‘first order logic’:
  \[ \{ t \mid P(t) \} \]
  
  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 TRC Queries

- The allowed symbols:
  \[ \land, \lor, \lnot, \Rightarrow, \geq, \leq, \neq, \nabla, \left\langle, \left\rangle, \in \]

- Quantifiers:
  \[ \forall, \exists \]
Syntax of TRC Queries

- ‘Atomic formulas’:

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

Where \textit{op} is an operator in the set \{<, >, =, \leq, \geq, \neq\}
Syntax of TRC Queries

- A ‘formula’ is:
  - Any atomic formula

- If $P1$ and $P2$ are formulas, so are
  
  $\neg P1; \neg P2; P1 \land P2; P1 \lor P2; P1 \Rightarrow P2$

- 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></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ssn</strong></td>
<td><strong>Name</strong></td>
<td><strong>Address</strong></td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>QF ave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c-id</strong></td>
<td><strong>c-name</strong></td>
<td><strong>Units</strong></td>
</tr>
<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>

<table>
<thead>
<tr>
<th>TAKES</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSN</strong></td>
<td><strong>c-id</strong></td>
<td><strong>Grade</strong></td>
</tr>
<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 all student records

\( \{ t \mid t \in \text{STUDENT} \} \)

Output tuple of type ‘STUDENT’
Examples

- Find the student record with ssn=123
Examples

- Find the student record with $ssn=123$

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

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

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

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

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 part time or 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>

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

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

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

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

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>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>QF ave</td>
</tr>
</tbody>
</table>

<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>

<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>

2-way Join!
More Examples

- Find the names of students taking 15-415

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

- Find the 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 \} \]

join

projection

selection
More Examples

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

<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>QF ave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c-id</td>
<td>c-name</td>
<td>units</td>
<td></td>
</tr>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>c-id</td>
<td>grade</td>
</tr>
<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

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

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></th>
<th>PC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p-id</td>
<td>c-id</td>
<td>p-id</td>
<td>c-id</td>
</tr>
<tr>
<td>Mary</td>
<td>Tom</td>
<td>Mary</td>
<td>Tom</td>
</tr>
<tr>
<td>Peter</td>
<td>Mary</td>
<td>Peter</td>
<td>Mary</td>
</tr>
<tr>
<td>John</td>
<td>Tom</td>
<td></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>
</tr>
</thead>
<tbody>
<tr>
<td>s#</td>
<td>p#</td>
</tr>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p1</td>
</tr>
<tr>
<td>s5</td>
<td>p3</td>
</tr>
</tbody>
</table>

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

- Find suppliers that shipped all the bad parts

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

What is the equivalence of this in Relational Algebra?
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 that take all the courses that ssn=123 does (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 that take all the courses that ssn=123 does (and maybe even more)

\[ \{o \mid \forall t ((t \in TAKES \land t.\text{ssn} = 123) \Rightarrow \exists t_1 \in TAKES (t_1.\text{c} - id = t.\text{c} - id \land t_1.\text{ssn} = o.\text{ssn}) \} \]
‘Proof’ of Equivalence

- Relational Algebra <-> TRC

But...
Safety of Expressions

- What about?
  \[ \{ t \mid t \not\in \text{STUDENT} \} \]
  It has infinite output!!

- Instead, always use:
  \[ \{ t \mid \ldots t \in \text{SOME \text{-} TABLE} \} \]
Outline

- The Division Operator and Summary of Relational Operators
- Tuple Relational Calculus
- Domain Relational Calculus
Overview - Detailed

- Domain Relational Calculus (DRC)
  - Why?
  - Details
  - Examples
  - Equivalence with TRC and relational algebra
  - ‘Safety’ of expressions
Domain Relational Calculus (DRC)

- **Question**: why?

- **Answer**: slightly easier than TRC, although equivalent basis for QBE

- **Idea**: “domain” variables instead of “tuple” variables

- **Example**: ‘find STUDENT record with ssn=123’

\[
\{< s,n,a > |< s,n,a > \in STUDENT \land s = 123\}
\]
Syntax of DRC Queries

- The allowed symbols are:

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

- Quantifiers:

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

- But: domain (= column) variables, as opposed to tuple variables:

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

\[
\begin{align*}
    \text{ssn} & \quad \text{name} & \quad \text{address} \\
\end{align*}
\]
Reminder: Our Mini University DB

<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>QF ave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c-id</td>
<td>c-name</td>
<td>units</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>TAKES</th>
<th></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 all student records

\[ \{< s, n, a > | < s, n, a > \in STUDENT \} \]

- What is the equivalence of this in TRC?

\[ \{ t | t \in STUDENT \} \]
Examples

- Find the student record with ssn=123

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

OR:

\[ \{ <123, n, a> | <123, n, a> \in STUDENT \} \]

In TRC:

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

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

- Find the name of student with ssn=123

\[ \{ <n> | \quad <123,n,a> \in STUDENT \} \]

**In TRC:** \[ \{ t | \exists s \in STUDENT (s.ssn = 123 \wedge t.name = s.name) \} \]
Examples

- Find the name of student with ssn=123

\{ <n> | \exists a(<123,n,a> \in STUDENT) \} \\
\uparrow \text{need to ‘bind’ ‘a’}

\textbf{In TRC:} \quad \{ t | \exists s \in STUDENT (s.ssn = 123 \land \\
\quad t.name = s.name) \}

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

- Get records of both PT and FT students

\[
\{< s, n, a > | < s, n, a > \in FT\_STUDENT \lor < s, n, a > \in PT\_STUDENT \}\n\]

In TRC: \[ \{ t | t \in FT\_STUDENT \lor t \in PT\_STUDENT \}\]

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

- Find the students that are not staff

\[
\{< s, n, a > | < s, n, a > \in \text{STUDENT} \land < s, n, a > \not\in \text{STAFF}\}
\]

In TRC: \[
\{ t | t \in \text{STUDENT} \land t \not\in \text{STAFF}\}
\]

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

- Find all the pairs of (male, female)

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

In TRC: \{ t \mid \exists m \in MALE \land \\
\exists f \in FEMALE \\
(t.m - name = m.name \land \\
t.f - name = f.name) \}

This is equivalent to the ‘Cartesian Product’ operator in Relational Algebra!
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>Name</td>
<td>c-name</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
</tr>
<tr>
<td>---------</td>
<td>-------</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>QF ave</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
<th>Rodríguez</th>
<th>15-413</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>c-id</td>
<td>grade</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

2-way Join!
Examples

- Find the names of students taking 15-415

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

In TRC: \[
\{ t | \exists s \in STUDENT \\
\land \exists e \in Takes (s.ssn = e.ssn \land \\
t.name = s.name \land \\
e.c - id = 15-415) \}
\]

This is equivalent to the ‘Join’ operator in Relational Algebra!
A Sneak Preview of QBE

- Very user friendly
- Heavily based on RDC
- Very similar to MS Access interface

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

<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>
More Examples

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

<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>Name</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>s.e.</td>
</tr>
<tr>
<td>jones</td>
<td>o.s.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>main str</td>
<td>2</td>
</tr>
<tr>
<td>QF ave</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123</td>
<td>15-413</td>
<td>A</td>
</tr>
<tr>
<td></td>
<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

In TRC:

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

join

projection

selection
More Examples

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

**In DRC:**

\{
\langle n \rangle \mid \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
\langle s, n, a \rangle \in STUDENT \wedge \\
\langle s, c, g \rangle \in TAKES \wedge \\
\langle c, cn, 2 \rangle \in CLASS \}

More Examples

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

\[
\{ < n > | \exists s, a, c, g, cn( < s, n, a > \in STUDENT \land < s, c, g > \in TAKES \land < c, cn, 2 > \in CLASS ) \}
\]

In DRC:

Easier than TRC!
Even More Examples

- Find Tom’s grandparent(s)

<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>

<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>

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

**In DRC:**
\[
\{ < g > \mid \exists p(< g, p > \in PC \land \\
< p,"Tom" > \in PC ) \}
\]
Harder Examples: DIVISION

- Find suppliers that shipped all the bad parts
Harder Examples: DIVISION

• Find suppliers that shipped all the bad parts

**In TRC:**

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

**In DRC:**

\[
\{ < s > \mid \forall p (< p > \in BAD \_ P \Rightarrow < s, p > \in SHIPMENT) \}
\]
More on Division

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

In TRC:

\[ \{ o \mid \forall t((t \in \text{TAKES} \land t.\text{ssn} = 123) \Rightarrow \exists t1 \in \text{TAKES} ( t1.c - \text{id} = t.c - \text{id} \land t1.\text{ssn} = o.\text{ssn}) \} \} \]
More on Division

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

**In DRC:**

\[ \{ <s > | \forall c (\exists g (<123, c, g > \in TAKES) \Rightarrow \exists g'(<s, c, g'>) \in TAKES) ) \} \]
‘Proof’ of Equivalence

- Relational Algebra <-> Domain Relational Calculus <-> Tuple Relational Calculus

But...
Safety of Expressions

- Similar to TRC

- FORBIDDEN:

\[ \{< s,n,a >|< s,n,a > \notin STUDENT \} \]
Summary

- The relational model has rigorously defined query languages — simple and powerful

- Relational algebra is more operational/procedural
  - Useful for internal representation of query evaluation plans

- Relational calculus is declarative
  - Users define queries in terms of what they want, not in terms of how to compute them
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