**10/5/2021**

**Introduction to Relational Calculus**

by K. Yue

**1. Introduction**

* Non-procedural, declarative, and high level.
* Two kinds:
  1. Domain Relational Calculus (DRC)
  2. Tuple Relational Calculus (TRC)
* *Queries* specified by the set builder form: {s | cond(s) }
* cond(s) is known as a *formula*.
* Constructs:
  1. Variables:
     1. TRC: tuples (bound to tuples): e.g. s, t, student, class, etc.
     2. DRC: attributes (bound to domain value): e.g. a, b, c, stuId, firstName, etc.
     3. The variables are sometime known as 'dummy variables.' They can assume any names.
  2. Constants: string, int, etc. E.g. 12, 'csci', 3.7.
  3. Comparison operators: <, <, =, etc.
  4. Boolean operators: and (conjunction, ∧ or just ,), or (disjunction ∨), not (¬), implies (⇒), etc.
  5. Membership functions: belongs to, ∈, not belongs to, ∉, etc.
  6. Quantifiers: there exists (existential, ∃), not exists (∄), for all (universal ∀).
* An *atom* can be thought of as a simple Boolean expression: x op y, where x and y are attributes or constants, and op is a comparison operation.
  1. Example: R.a>=2, sname='BBC', t ∉ T, etc.
* A *formula* is either an atom or formulas connected by Boolean operators or qualifiers.
  1. Example: ∃e(e ∈ R), (and) a=1 ∨ (or) b=2, ∃a,b,c((a,b,c) ∈ R) or simply (a,b,c) ∈ R
* A formula that is not an atom can be thought of a compound Boolean expression.
* A variable is *bound* if it has a finite number of values., such as appearing in qualifier expressions (without negation). A *free* variable is not bound.
* Relational Calculus expressions need to be *safe*: results should be a finite set of tuples.
* Care should be taken especially for the negation operation. E.g. {s |¬ (s ∈ Student) } is unsafe.
* For a given implementation of relational calculus:
  1. There may be restrictions of supported constructs.
  2. There may be certain *canonical* requirements: e.g. conjunction (and operator joined: e.g. a and b; (aVb) and (-aVc), etc.) of disjunction (joined by or). PROLOG.
* Relational Calculus and Relational Algebra:
  1. All RA expressions can be expressed in RC.
  2. RA and RC have the same expressive power.
  3. Any query language that can express all RA queries is known to be relational complete.

**Example:**

{i | i ∈ I ∧ i % 2 =0}  
{i | i ∈ I, i % 2 =0} -- set builder form.

{t | ∃r ∈R, r.firstname = t.firstname, r.lastname = t.lastname}

* t is a free variable.
* It will have two attributes: t.firstname and t.lastname.

Alternatively, we can use the set builder form in the LHS before |:

{(r.firstname, r.lastname) | r ∈ R}

R(A,B,C,D) / S(C,D)

{(a,b) | ∀(c,d) ∈ S((a,b,c,d) ∈ R))}

* In this class, with the query is complicated, you may use intermediate relations to construct a sequence of RC expressions to provide the result.

**Exercises:**

How do you use RC to implement RA operations?

**2. TRC**

* The variables in TORC are tuples.
* SQL is based on TRC.

**Exercise:**

Work on some of the query questions listed in the [Supplies database example](http://dcm.uhcl.edu/yue/courses/csci5333/current/notes/Query/QueryExercise.pdf) in TRC.

**3. DRC**

* The variables in DRC are attributes (domain values).
* May use anonymous variable: \_ as a placeholder.
* Query By Example (QBE) is based on DRC.

**Exercise:**

Work on some of the query questions listed in the [Supplies database example](http://dcm.uhcl.edu/yue/courses/csci5333/current/notes/Query/QueryExercise.pdf) in DRC.

3. Show all information of suppliers with a status greater than 5.

|  |  |  |  |
| --- | --- | --- | --- |
| **SNum** | **SName** | **SCity** | **Status** |
| S1 | ABC | Dallas | 10 |
| S2 | DEF | Houston | 20 |
| S3 | Go go | Houston | 12 |

**SELECT** s.\*

**FROM** supplier **AS** s -- s ∈ supplier,

**WHERE** s.**status** > 5;

TRC: s: a tuple/row (SQL uses TRC)

{s | s ∈ supplier, s.status > 5}

DRC: (e.g. Access; Prolog, …)

(): tuple builder

{(snum, sname, scity, status) | (snum, sname, scity, status) ∈ supplier, status > 5}

{(snum, sname, scity, status) | (snum, sname, scity, status) ∈ supplier, status\_1 > 5} incorrect

{(a,b,c,d) | (a,b,c,d) ∈ supplier and d > 5}

5. Show all information of parts with a color of Red or Blue.

TRC:

{ p | p ∈ part, (p.color = ‘Red’ V p.color = ‘Blue’)}

p ∈ part, (p.color = ‘Red’ V p.color = ‘Blue’)}: (p ∈ part) and (p.color = ‘Red’ V p.color = ‘Blue’)}: conjunction of two disjunctions.

DRC:

{ (pnum, pname, color, weight) | (pnum, pname, color, weight) ∈ part, (color = ‘Red’ V color = ‘Blue’)}

16. Show all information of parts supplied by supplier S2.

|  |
| --- |
| P1 |
| P2 |
| P4 |

Part

|  |  |  |  |
| --- | --- | --- | --- |
| **PNum** | **PName** | **Color** | **Weight** |
| P1 | Drum | Green | 10 |
| P2 | Hammer | Green | 20 |
| P4 | Micropod | Red | 4 |

Supply:

|  |  |
| --- | --- |
| **SNum** | **PNum** |
| S2 | P1 |
| S2 | P2 |
| S2 | P4 |

TRC:

{ p | p ∈ part, s ∈ supplier, s.snum = ‘S2’, s.pnum = p.pnum)}

**SELECT** **DISTINCT** p.\*

**FROM** part **AS** p **INNER** **JOIN** supply **AS** s **ON** (p.pnum = s.PNUM)

-– alias: p: row in part

**WHERE** s.SNUM = 'S2';

DRC:

{ { (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (snum, pnum, quantity) ∈ supply, snum = ‘S2’)}

{ { (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (snum, pnum, \_) ∈ supply, snum = ‘S2’)}

\_: placeholder, anon variable.

{ (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (‘S2’, pnum, \_) ∈ supply)}

16b. Show pnum and pname of parts supplied by supplier S2.

TRC:

{ (p.pnum, p.pname) | p ∈ part, s ∈ supply, s.snum = ‘S2’, s.pnum = p.pnum)}

**SELECT** **DISTINCT** p.pnum, p.pname

**FROM** part **AS** p **INNER** **JOIN** supply **AS** s **ON** (p.pnum = s.PNUM)

**WHERE** s.SNUM = 'S2';

DRC:

{ { (pnum, pname) | { (pnum, pname, \_, \_) ∈ part, (snum, pnum, \_) ∈ supply, snum = ‘S2’)}

F(x) = x2

F(yue) = yue2

Dummy variable

17. Show all information of parts supplied by supplier S2 or S3.

TRC:

{ p | p ∈ part, s ∈ supply, (s.snum = ‘S2’ V s.snum= ‘S3’), s.pnum = p.pnum)}

DRC:

{ { (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (snum, pnum, \_) ∈ supply, (snum = ‘S2’ V snum=’S3’)}

19. Show all information of parts supplied by supplier S2 and S3.

|  |  |  |  |
| --- | --- | --- | --- |
| S1: | S2 | P4 | 6 |
| S2: | S3 | P4 | 1 |

TRC:

{ p | p ∈ part, s ∈ supply, s.snum = ‘S2’ , s.snum= ‘S3’, s.pnum = p.pnum)} = empty set.

X == 1 && x==2

Show all information of parts supplied by supplier S2 and S3.

{ p | p ∈ part, s1 ∈ supplier, s1.snum = ‘S2’ , s1.pnum = p.pnum, , s2 ∈ supplier, s2.snum = ‘S3’ , s2.pnum = p.pnum)}

**SELECT** **DISTINCT** p.pnum, p.pname

**FROM** part **AS** p **INNER** **JOIN** supply **AS** s1 **ON** (p.pnum = s1.PNUM)

**INNER** **JOIN** supply **AS** s2 **ON** (p.pnum = s2.PNUM)

**WHERE** s1.SNUM = 'S2'

**AND** s2.SNUM = 'S3';

Show all information of parts supplied by supplier S2 and S3.

DRC:

{ { (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (‘S2’, pnum, \_) ∈ supply, , (‘S3’, pnum, \_) ∈ supply)}

19b. Show all information of parts supplied by supplier S2 but not S3.

{ p | p ∈ part, s ∈ supply, s.snum = ‘S2’, s.snum<>’S3’ (subsumed), s.pnum = p.pnum)}

The same answer as:

{ p | p ∈ part, s ∈ supply, s.snum = ‘S2’, s.pnum = p.pnum)}

Because s.snum = ‘S2’ => s.snum<>’S3’ (vs. x==1 => x <> 2)

|  |  |  |
| --- | --- | --- |
| **SNum** | **PNum** | **Quantity** |
| S2 | P1 | 11 |
| S2 | P2 | 1 |
| S2 | P4 | 6 |
| S3 | P4 | 1 |
| S3 | P5 | 2 |
| S3 | P6 | 12 |
| S3 | P7 | 5 |

{ p | p ∈ part, s ∈ supply, s.snum = ‘S2’, ((s2 ∈ supply, s2.pnum = p.pnum) => s2.snum <> ‘S3’) s.pnum = p.pnum)}

(s2 ∈ supply, s2.pnum = p.pnum) => s2.snum <> ‘S3’ not a disjunction clause.

p => q equivalent -p V q

-(p and q) equivalent to -p V -q

(s2 ∈ supply, s2.pnum = p.pnum) => s2.snum <> ‘S3’:

-(s2 ∈ supply, s2.pnum = p.pnum) V s2.snum <> ‘S3’

-s2 ∈ supply V -s2.pnum = p.pnum V s2.snum <> ‘S3’

s2 ∉ supply V s2.pnum <> p.pnum V s2.snum <> ‘S3’

{ p | p ∈ part, s ∈ supply, s.snum = ‘S2’, (s2 ∉ supply V s2.pnum <> p.pnum V s2.snum <> ‘S3’), s.pnum = p.pnum)}

SQL:

**SELECT** **DISTINCT** p.pnum, p.pname

**FROM** part **AS** p **INNER** **JOIN** supply **AS** s1 **ON** (p.pnum = s1.PNUM)

**WHERE** s1.SNUM = 'S2'

**AND** **NOT** **EXISTS**

(**SELECT** s2.\*

**FROM** supply **AS** s2

**WHERE** s2.PNUM = p.pnum

**AND** s2.SNUM = 'S3');

DRC:

{ (pnum, pname, color, weight) | { (pnum, pname, color, weight) ∈ part, (‘S2’, pnum, \_) ∈ supply, , (‘S3’, pnum, \_) ∉ supply)}

It is possible for R(A,B,C,D,E) to have exactly three superkeys.

CK: AB => SK: AB, ABC, ABD, ABE, ABCD, ABCE, ABDE, ABCDE  
CK: ABC => SK: ABC, ABCD, ABCE, ABCDE  
CK: ABCD => SK: ABCD, ABCDE  
CK: ABCDE => SK: ABCDE

CK: [1] ABCD, [2] ABCE => SK: ABCD, ABCE, ABCDE.

SQL:

17. Show all information of parts supplied by supplier S2 or S3. Ordered by descending weight

Output columns:

1. p.\*

Source tables:

1. part AS p
2. supply as s

Join conditions:

1. p.pnum = s.snum

Problem Conditions: supplied by supplier S2 or S3.

1. S.snum = ‘S2’ or s.snum = ‘S3’

Viewing Orders:

1. ORDEY BY p.weight DESC