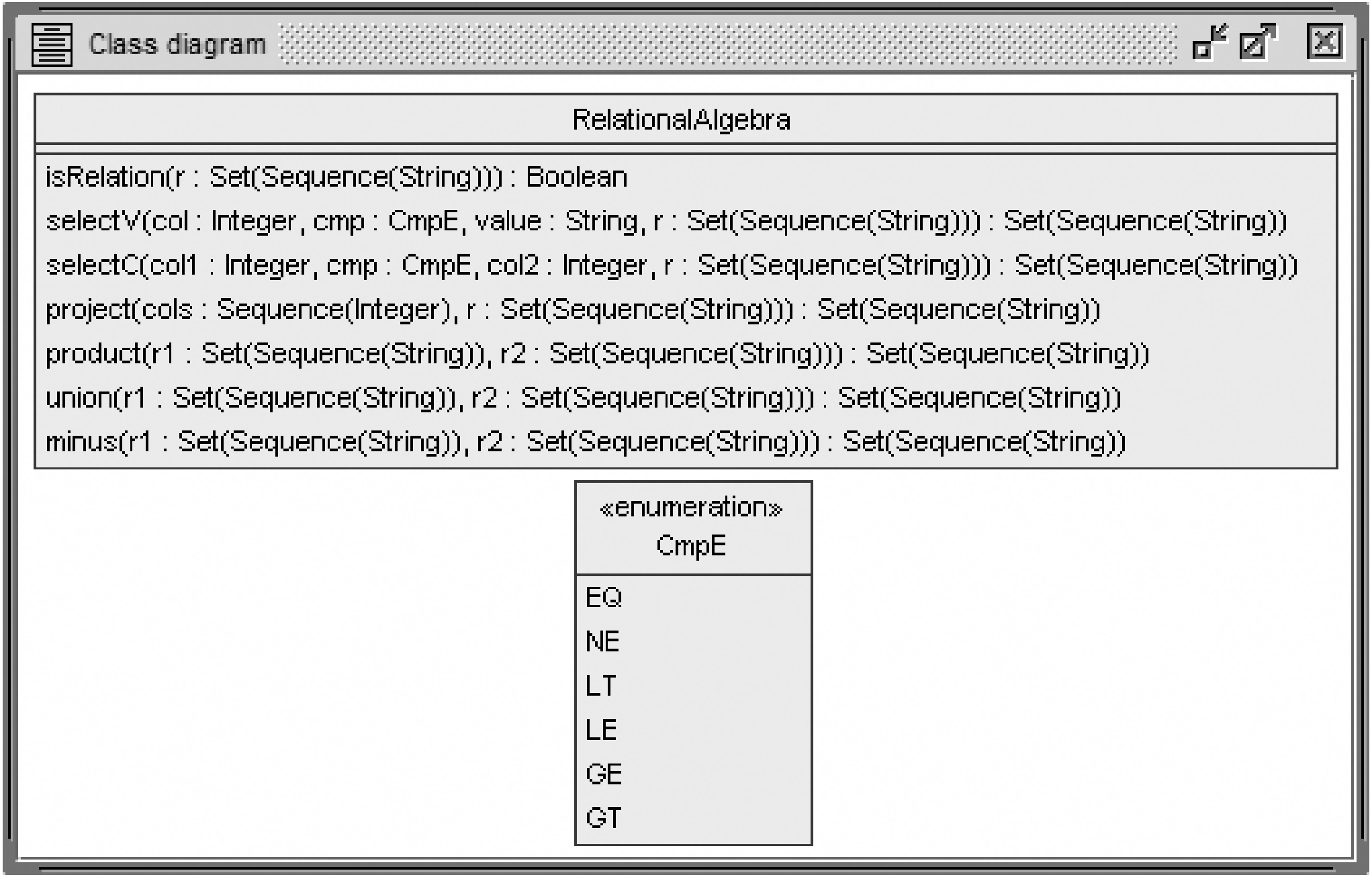
A Quick and Dirty Realization of Relational Algebra in an Object-Oriented Design Tool

martin gogolla

2016-11-18

- Realization of Relational Algebra within the design tool USE
- Representation of a relation as a term of type Set(Sequence(String))
- Singleton class RelationalAlgebra
- Possessing operations for selection, projection, product, union and difference of relations that establish the Relational Algebra
- Operations realized as query operations with OCL expressions


Class diagram
RelationalAlgebra
isRelation(r : Set(Sequence(String))) : Boolean
selectV(col : Integer, cmp : CmpE, value : String, r : Set(Sequence(String))) : Set(Sequence(String))
selectC(col1 : Integer, cmp : CmpE, col2 : Integer, r : Set(Sequence(String))) : Set(Sequence(String))
project(cols : Sequence(Integer), r : Set(Sequence(String))) : Set(Sequence(String))
product(r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
union(r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
minus(r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
«enumeration»
CmpE
EQ
NE
LT
LE
GE
GT

$\sigma_{col\ cmp\ value}(r)$	`selectV(col:Integer,`
	`  cmp:CmpE,`
	`  value:String,`
	`  r:Set(Seq(String))) : Set(Seq(String))`

$\sigma_{col1\ cmp\ col2}(r)$	`selectC(col1:Integer,`
	`  cmp:CmpE,`
	`  col2:Integer,`
	`  r:Set(Seq(String))) : Set(Seq(String))`

$\pi_{cols}(r)$	`project(cols:Seq(Integer),` `  r:Set(Seq(String))) : Set(Seq(String))`
r1 x r2	`product(r1:Set(Seq(String)),` `  r2:Set(Seq(String))) : Set(Seq(String))`
r1 ∪ r1	`union(r1:Set(Seq(String)),` `  r2:Set(Seq(String))) : Set(Seq(String))`
r1 - r2	`minus(r1:Set(Seq(String)),` `  r2:Set(Seq(String))) : Set(Seq(String))`

```
isRelation(r:Set(Sequence(String))):Boolean=
  r->forAll(t1,t2|t1->size()=t2->size())

Assumptions
- r, r1, r2 from below satisfy predicate isRelation
- col, col1, col2 are single relation columns
- cols, cols1, cols2 are sequences of relation columns

selectV(col:Integer, cmp:CmpE, value:String,
  r:Set(Sequence(String))):Set(Sequence(String))=
    if cmp=#EQ then r->select(t|t->at(col)= value) else
    if cmp=#NE then r->select(t|t->at(col)<>value) else
    if cmp=#LT then r->select(t|t->at(col)< value) else
    if cmp=#LE then r->select(t|t->at(col)<=value) else
    if cmp=#GE then r->select(t|t->at(col)>=value) else
    if cmp=#GT then r->select(t|t->at(col)> value) else
      Set{}
    endif endif endif endif endif endif

selectC(col1:Integer, cmp:CmpE, col2:Integer,
  r:Set(Sequence(String))):Set(Sequence(String))=
    if cmp=#EQ then r->select(t|t->at(col1)= t->at(col2)) else
    if cmp=#NE then r->select(t|t->at(col1)<>t->at(col2)) else
    if cmp=#LT then r->select(t|t->at(col1)< t->at(col2)) else
    if cmp=#LE then r->select(t|t->at(col1)<=t->at(col2)) else
    if cmp=#GE then r->select(t|t->at(col1)>=t->at(col2)) else
    if cmp=#GT then r->select(t|t->at(col1)> t->at(col2)) else
      Set{}
    endif endif endif endif endif endif
```

```
project(cols:Sequence(Integer), r:Set(Sequence(String))):
  Set(Sequence(String))=
    r->iterate(t1;
      res1:Set(Sequence(String))=Set{}|
        let t2=cols->iterate(i;
           res2:Sequence(String)=Set{}|
             res2->including(t1->at(i))) in
               res1->including(t2))

product(r1:Set(Sequence(String)), r2:Set(Sequence(String))):
  Set(Sequence(String))=
    r1->iterate(t1;
      res1:Set(Sequence(String))=Set{}|
        r2->iterate(t2;
          res2:Set(Sequence(String))=res1|
            res2->including(t1->union(t2))))

union(r1:Set(Sequence(String)), r2:Set(Sequence(String))):
  Set(Sequence(String))=r1->union(r2)

minus(r1:Set(Sequence(String)), r2:Set(Sequence(String))):
  Set(Sequence(String))=r1-r2
```

```
Country | CName          | Capital     | CPop
--------+----------------+-------------+------
        | 'Germany',     | 'Berlin',   | '80'
        | 'France',      | 'Paris',    | '60'
        | 'Netherlands' | 'Amsterdam' | '25'

Town | TName        | TPop
-----+--------------+------
     | 'Berlin'     | '4'
     | 'Hamburg'    | '2'
     | 'Koeln'      | '1'
     | 'Paris'      | '9'
     | 'Marseille'  | '2'
     | 'Amsterdam'  | '2'
```

```
Country=Set{Sequence{'Germany',    'Berlin',   '80'},
            Sequence{'France',     'Paris',    '60'},
            Sequence{'Netherlands','Amsterdam','25'}}

Town=Set{Sequence{'Berlin',   '4'},
         Sequence{'Hamburg',  '2'},
         Sequence{'Koeln',    '1'},
         Sequence{'Paris',    '9'},
         Sequence{'Marseille','2'},
         Sequence{'Amsterdam','2'}}
```

$\sigma_{CPop \leq '60'}(Country)$

Country tuples where the
country population is less equal to 60

```
ra.selectV(3,#LE,'60',Country)
```

$\sigma_{CName > Capital}(Country)$

Country tuples where the country
name comes after the capital name

```
ra.selectC(1,#GT,2,Country)
```

```
Country | CName | Capital | CPop
--------+-------+---------+------

Town | TName | TPop
-----+-------+------
```

$\pi_{Cpop,CName}(Country)$

```
ra.project(Sequence{3,1},Country)
```

From Country tuples only
the population and name

```
Country X Town
```

```
ra.product(Country,Town)
```

Cross product of
Country and Town

Country name together with capital name and capital population

$\pi_{CName,Capital,TPop}(\sigma_{Capital=TName}(Country \times Town))$

```
ra.project(Sequence{1,2,5},
  ra.selectC(2,#EQ,4,
    ra.product(Country,Town)))
```

```
Country | CName | Capital | CPop
--------+-------+---------+------

Town | TName | TPop
-----+-------+------
```

select CName, Capital, TPop
from Country, Town
where Capital=TName

Names of towns not being capitals

$\pi_{TName}(Town) - \pi_{Capital}(Country)$

```
ra.minus(ra.project(Sequence{1},Town),
         ra.project(Sequence{2},Country))
```

Names of geographical units (town or country) together with the unit's population

$Town \cup \pi_{CName,CPop}(Country)$

```
ra.union(Town,ra.project(Sequence{1,3},Country))
```

```
use> !create ra:RelationalAlgebra

use> \
     !let Country=Set{Sequence{'Germany',    'Berlin',   '80'},
                      Sequence{'France',     'Paris',    '60'},
                      Sequence{'Netherlands','Amsterdam','25'}}
.

use> ?ra.isRelation(Country)
     true : Boolean

use> \
     !let Town=Set{Sequence{'Berlin',   '4'},
                   Sequence{'Hamburg',  '2'},
                   Sequence{'Koeln',    '1'},
                   Sequence{'Paris',    '9'},
                   Sequence{'Marseille','2'},
                   Sequence{'Amsterdam','2'}}
.

use> ?ra.isRelation(Town)
     true : Boolean

use> ?ra.isRelation(Set{Sequence{'Paris','9'},Sequence{'France','Wine','Bad'}})
     false : Boolean
```

Country tuples where the
country population is less equal to 60

```
use> ?ra.selectV(3,#LE,'60',Country)
     Set{Sequence{'France',     'Paris',    '60'},
         Sequence{'Netherlands','Amsterdam','25'}} : Set(Sequence(String))
```

Country tuples where the country
name comes after the capital name

```
use> ?ra.selectC(1,#GT,2,Country)
     Set{Sequence{'Germany',    'Berlin',   '80'},
         Sequence{'Netherlands','Amsterdam','25'}} : Set(Sequence(String))
```

From Country tuples only
the population and name

```
use> ?ra.project(Sequence{3,1},Country)
     Set{Sequence{'25','Netherlands'},
         Sequence{'60','France'     },
         Sequence{'80','Germany'}   } : Set(Sequence(String))
```

Cross product of
Country and Town

```
use> ?ra.product(Country,Town)
     Set{Sequence{'France',      'Paris',     '60', 'Amsterdam', '2'},
         Sequence{'France',      'Paris',     '60', 'Berlin',    '4'},
         Sequence{'France',      'Paris',     '60', 'Hamburg',   '2'},
         Sequence{'France',      'Paris',     '60', 'Koeln',     '1'},
         Sequence{'France',      'Paris',     '60', 'Marseille', '2'},
         Sequence{'France',      'Paris',     '60', 'Paris',     '9'},
         Sequence{'Germany',     'Berlin',    '80', 'Amsterdam', '2'},
         Sequence{'Germany',     'Berlin',    '80', 'Berlin',    '4'},
         Sequence{'Germany',     'Berlin',    '80', 'Hamburg',   '2'},
         Sequence{'Germany',     'Berlin',    '80', 'Koeln',     '1'},
         Sequence{'Germany',     'Berlin',    '80', 'Marseille', '2'},
         Sequence{'Germany',     'Berlin',    '80', 'Paris',     '9'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Amsterdam', '2'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Berlin',    '4'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Hamburg',   '2'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Koeln',     '1'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Marseille', '2'},
         Sequence{'Netherlands', 'Amsterdam', '25', 'Paris',     '9'}} :
         Set(Sequence(String))
```

```
use> \
?ra.project(Sequence{1,2,5},
        ra.selectC(2,#EQ,4,
          ra.product(Country,Town)))
.
     Set{Sequence{'France',      'Paris',     '9'},
         Sequence{'Germany',     'Berlin',    '4'},
         Sequence{'Netherlands', 'Amsterdam', '2'}} : Set(Sequence(String))
```

Country name together with capital name and capital population

```
use> \
     ?ra.minus(ra.project(Sequence{1},Town),
               ra.project(Sequence{2},Country))
.
     Set{Sequence{'Hamburg'  },
         Sequence{'Koeln'    },
         Sequence{'Marseille'}} : Set(Sequence(String))
```

Names of towns not being capitals

```
use> ?ra.union(Town,ra.project(Sequence{1,3},Country))
     Set{Sequence{'Amsterdam',    '2'},
         Sequence{'Berlin',       '4'},
         Sequence{'France',      '60'},
         Sequence{'Germany',     '80'},
         Sequence{'Hamburg',      '2'},
         Sequence{'Koeln',        '1'},
         Sequence{'Marseille',    '2'},
         Sequence{'Netherlands', '25'},
         Sequence{'Paris',        '9'}} : Set(Sequence(String))
```

Names of geographical units (town or country) together with the unit's population

Thanks!