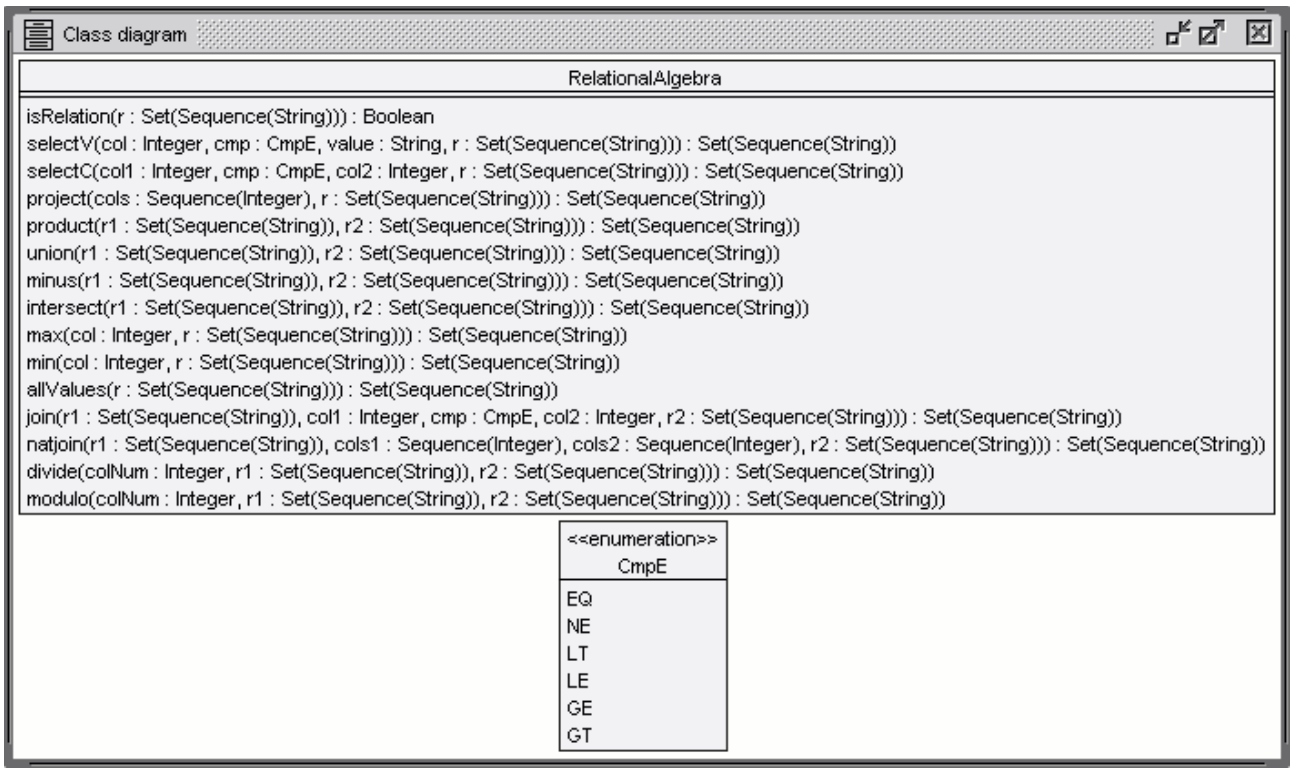
Class diagram
RelationalAlgebra
isRelation(r : Set(Sequence(String))) : Boolean
selectV(col : Integer, cmp : CmpE, value : String, r : Set(Sequence(String))) : Set(Sequence(String))
selectC(col1 : Integer, cmp : CmpE, col2 : Integer, r : Set(Sequence(String))) : Set(Sequence(String))
project(cols : Sequence(Integer), 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))
intersect(r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
max(col : Integer, r : Set(Sequence(String))) : Set(Sequence(String))
min(col : Integer, r : Set(Sequence(String))) : Set(Sequence(String))
allValues(r : Set(Sequence(String))) : Set(Sequence(String))
join(r1 : Set(Sequence(String)), col1 : Integer, cmp : CmpE, col2 : Integer, r2 : Set(Sequence(String))) : Set(Sequence(String))
natjoin(r1 : Set(Sequence(String)), cols1 : Sequence(Integer), cols2 : Sequence(Integer), r2 : Set(Sequence(String))) : Set(Sequence(String))
divide(colNum : Integer, r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
modulo(colNum : Integer, r1 : Set(Sequence(String)), r2 : Set(Sequence(String))) : Set(Sequence(String))
<<enumeration>>
CmpE
EQ
NE
LT
LE
GE
GT

```
model RelationalAlgebraWorld

enum CmpE {EQ, NE, LT, LE, GE, GT}

class RelationalAlgebra
operations
  ------------------------------------------------------------------------
  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
      oclEmpty(Set(Sequence(String)))
    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
      oclEmpty(Set(Sequence(String)))
    endif endif endif endif endif endif
  ------------------------------------------------------------------------
  project(cols:Sequence(Integer), r:Set(Sequence(String))):
    Set(Sequence(String))=
    r->iterate(t1;
      res1:Set(Sequence(String))=oclEmpty(Set(Sequence(String)))|
      let t2=cols->iterate(i;
        res2:Sequence(String)=oclEmpty(Sequence(String))|
        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))=oclEmpty(Set(Sequence(String)))|
      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
  ------------------------------------------------------------------------
```

```
  intersect(r1:Set(Sequence(String)), r2:Set(Sequence(String))):
    Set(Sequence(String))=minus(r1,minus(r1,r2))
  ------------------------------------------------------------------------
  max(col:Integer, r:Set(Sequence(String))):Set(Sequence(String))=
    if r->isEmpty() then oclEmpty(Set(Sequence(String))) else
      minus(r,project(Sequence{1..r->any(true)->size()},
        selectC(col,#LT,r->any(true)->size()+col,product(r,r))))
    endif
  ------------------------------------------------------------------------
  min(col:Integer, r:Set(Sequence(String))):Set(Sequence(String))=
    if r->isEmpty() then oclEmpty(Set(Sequence(String))) else
      minus(r,project(Sequence{1..r->any(true)->size()},
        selectC(col,#GT,r->any(true)->size()+col,product(r,r))))
    endif
  ------------------------------------------------------------------------
  allValues(r:Set(Sequence(String))):Set(Sequence(String))=
    if r->isEmpty() then oclEmpty(Set(Sequence(String))) else
      Set{1..r->any(true)->size()}->iterate(i;
        res:Set(Sequence(String))=oclEmpty(Set(Sequence(String)))|
        union(res,project(Sequence{i},r)))
    endif
  ------------------------------------------------------------------------
  join(r1:Set(Sequence(String)), col1:Integer, cmp:CmpE,
    col2:Integer, r2:Set(Sequence(String))):Set(Sequence(String))=
    if r1->isEmpty() then oclEmpty(Set(Sequence(String))) else
      selectC(col1,cmp,r1->any(true)->size()+col2,product(r1,r2))
    endif
  ------------------------------------------------------------------------
  -- assumption: cols1->size()=cols2->size()
  natjoin(r1:Set(Sequence(String)), cols1:Sequence(Integer),
    cols2:Sequence(Integer), r2:Set(Sequence(String))):
    Set(Sequence(String))=
    if r1->isEmpty() or r2->isEmpty() then
      oclEmpty(Set(Sequence(String)))
    else
      project(Sequence{1..r1->any(true)->size()}->
                union(cols2->iterate(i;
                  res:Sequence(Integer)=
                    Sequence{1..r2->any(true)->size()}|
                      res->excluding(i))->
                  collect(i|r1->any(true)->size()+i)),
        Set{1..cols1->size()}->iterate(i;
          res:Set(Sequence(String))=product(r1,r2)|
          selectC(cols1->at(i),#EQ,
            r1->any(true)->size()+cols2->at(i),res)))
    endif
  ------------------------------------------------------------------------
  -- assumption r1(1,...,colNum,colNum+1,...,r1ColNum)
  --                         r2(colNum+1,...,r1ColNum)
  divide(colNum:Integer, r1:Set(Sequence(String)),
    r2:Set(Sequence(String))):Set(Sequence(String))=
    let cols=Sequence{1..colNum} in
    minus(project(cols,r1),
          project(cols,minus(product(project(cols,r1),r2),r1)))
  ------------------------------------------------------------------------
  modulo(colNum:Integer, r1:Set(Sequence(String)),
    r2:Set(Sequence(String))):Set(Sequence(String))=
    minus(r1,product(divide(colNum,r1,r2),r2))
  ------------------------------------------------------------------------
end
```

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

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

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

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))
```

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

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))

use> ?ra.minus(ra.project(Sequence{1},Country),
        ra.project(Sequence{1},
          ra.selectC(3,#LT,6,ra.product(Country,Country))))
     Set{Sequence{'Germany'}} : Set(Sequence(String))

use> !let Job=Set{Sequence{'Ada','Analysis'},
                  Sequence{'Ada','Design'  },
                  Sequence{'Bea','Analysis'},
                  Sequence{'Bea','Design'  },
                  Sequence{'Bea','Coding'  },
                  Sequence{'Cyd','Design'  },
                  Sequence{'Cyd','Coding'  }}

use> -- R(A,B), S(B): project[A](R)-project[A](product(project[A](R),S)-R)

use> ?ra.minus(ra.project(Sequence{1},Job),
        ra.project(Sequence{1},
          ra.minus(ra.product(ra.project(Sequence{1},Job),
                              ra.project(Sequence{2},Job)),Job)))
     Set{Sequence{'Bea'}} : Set(Sequence(String))

use> ?ra.join(Country,2,#EQ,1,Town)
     Set{Sequence{'France',     'Paris',    '60','Paris',    '9'},
         Sequence{'Germany',    'Berlin',   '80','Berlin',   '4'},
         Sequence{'Netherlands','Amsterdam','25','Amsterdam','2'}} :
     Set(Sequence(String))

use> ?ra.project(Sequence{1,2,3,5},ra.join(Country,2,#EQ,1,Town))
     Set{Sequence{'France',     'Paris',    '60','9'},
         Sequence{'Germany',    'Berlin',   '80','4'},
         Sequence{'Netherlands','Amsterdam','25','2'}} : Set(Sequence(String))

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

use> ?let Age=Set{Sequence{'Ada','Apple', '42'},
                  Sequence{'Bea','Banana','36'},
                  Sequence{'Cyd','Cherry','42'},
                  Sequence{'Dan','Cherry','10'}} in
      let Hair=Set{Sequence{'Almond','Ada','Blonde'},
                   Sequence{'Banana','Bea','Black'},
                   Sequence{'Cherry','Can','Black'},
                   Sequence{'Cherry','Dan','Brown'}} in
     ra.natjoin(Age,Sequence{2,1},Sequence{1,2},Hair)
     Set{Sequence{'Bea','Banana','36','Black'},
         Sequence{'Dan','Cherry','10','Brown'}} : Set(Sequence(String))
```

```
use> ?let Rainy=Set{Sequence{'Oslo'},Sequence{'London'},Sequence{'Berlin'}} in
      let Sunny=Set{Sequence{'Berlin'},Sequence{'Rome'},Sequence{'Madrid'}} in
      ra.intersect(Rainy,Sunny)
      Set{Sequence{'Berlin'}} : Set(Sequence(String))

use> ?ra.max(3,Country)
     Set{Sequence{'Germany','Berlin','80'}} : Set(Sequence(String))

use> ?ra.min(3,Country)
     Set{Sequence{'Netherlands','Amsterdam','25'}} : Set(Sequence(String))

use> ?ra.max(2,Town)
     Set{Sequence{'Paris','9'}} : Set(Sequence(String))

use> ?ra.min(2,Town)
     Set{Sequence{'Koeln','1'}} : Set(Sequence(String))

use> ?ra.allValues(Country)
     Set{Sequence{'25'},
         Sequence{'60'},
         Sequence{'80'},
         Sequence{'Amsterdam'},
         Sequence{'Berlin'},
         Sequence{'France'},
         Sequence{'Germany'},
         Sequence{'Netherlands'},
         Sequence{'Paris'}} : Set(Sequence(String))

use> ?ra.allValues(Town)
     Set{Sequence{'1'},
         Sequence{'2'},
         Sequence{'4'},
         Sequence{'9'},
         Sequence{'Amsterdam'},
         Sequence{'Berlin'},
         Sequence{'Hamburg'},
         Sequence{'Koeln'},
         Sequence{'Marseille'},
         Sequence{'Paris'}} : Set(Sequence(String))

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

use> ?ra.divide(1,Job,ra.project(Sequence{2},Job))
     Set{Sequence{'Bea'}} : Set(Sequence(String))

use> !let Job=Set{Sequence{'Ada','Apple', 'UML', 'Analysis'},
                  Sequence{'Ada','Apple', 'UML', 'Design'},
                  Sequence{'Bea','Banana','UML', 'Analysis'},
                  Sequence{'Bea','Banana','UML', 'Design'},
                  Sequence{'Bea','Banana','Ruby','Coding'},
                  Sequence{'Cyd','Cherry','UML', 'Design'},
                  Sequence{'Cyd','Cherry','Ruby','Coding'}}

use> ?ra.divide(2,Job,ra.project(Sequence{3,4},Job))
     Set{Sequence{'Bea','Banana'}} : Set(Sequence(String))

use> ?ra.product(ra.divide(2,Job,ra.project(Sequence{3,4},Job)),
                 ra.project(Sequence{3,4},Job))
     Set{Sequence{'Bea','Banana','Ruby','Coding'},
         Sequence{'Bea','Banana','UML','Analysis'},
         Sequence{'Bea','Banana','UML','Design'}} : Set(Sequence(String))
```

```
use> ?ra.modulo(2,Job,ra.project(Sequence{3,4},Job))
     Set{Sequence{'Ada','Apple','UML','Analysis'},
         Sequence{'Ada','Apple','UML','Design'},
         Sequence{'Cyd','Cherry','Ruby','Coding'},
         Sequence{'Cyd','Cherry','UML','Design'}} : Set(Sequence(String))

use> ?ra.divide(3,Job,ra.project(Sequence{4},Job))
     Set{} : Set(Sequence(String))

use> ?ra.divide(2,ra.project(Sequence{1,2,4},Job),ra.project(Sequence{4},Job))
     Set{Sequence{'Bea','Banana'}} : Set(Sequence(String))
```