```
import itertools
from sets import Set
from SubGoal import SubGoal
from ConjQuery import ConjQuery
from itertools import chain, combinations
from MCD import MCD
from collections import defaultdict

def main():
    #Q,V = initializeQV()
    #Q,V = citingPapersSchema()
    Q,V = studentSchema()
    print "Query is : ",Q.toString
    print "Views are:"
    for v in V:
        print v.toString
    fd={}
    fd['P1']='S1'
    fd['Y1']='S1'
    print "Joint views formed are:"
    C = formMCDs(Q,V,fd)
    print "The MCDs constructed are :",len(C)
    for mcd in C:
        mcd.printMcd()

def createJointViews(V,fd):
    jointViews=[]
    viewPresent=False
    for k,v in fd.iteritems():
        for view in V:
            if v in view.head.args and k not in view.head.args:
                # a functional dependency is found in one of the
views, check all other views for possible joins
                for view1 in V:
                    if v in view1.head.args and view1.head.name !=
view.head.name:
                            # a pair from k in view and v in eachView
is found, both views can be joined
                            v3=joinView(view,view1)
                            # joint view added to same view
                            for pview in jointViews:
                                if pview.head.name ==
view1.head.name+view.head.name:
                                    viewPresent=True
                            if not viewPresent:
                                jointViews.append(v3)
    return jointViews
```

```
def joinView(v1,v2):

    args=[]
    args = v1.head.args + v2.head.args # each variable occuring twice
is only added once in the final list during this merge
    head= SubGoal(v1.head.name+v2.head.name, args)

    subgoals=[]
    subgoals = Set(v1.subgoals + v2.subgoals)
    return ConjQuery(head,subgoals)


def formMCDs(Q,setV,fd):
    C = []
    #print "Query is :",Q.toString
    jointViews = createJointViews(setV,fd)
    for v in jointViews:
            print v.toString
            C=C+formMCDsforView(Q,v)
    for V in setV:
        #print "View is :",V.toString
        C=C+formMCDsforView(Q,V)
    return C

def printSubsetSubGoal(subset):
    print "subset is = [ "+ " , ".join(s.toString for s in subset) +
"]"

def formMCDsforView(Q,V):
    C = []
    subsetsQ = chain.from_iterable(combinations(Q.subgoals, r) for r
in range(1,(len(Q.subgoals)+1)))
    for subsetQ in subsetsQ:
        if queryExists(C,subsetsQ):
            continue
        #printSubsetSubGoal(subsetQ)
        mcdlist=createmcd1(Q,V.mapQuery(),subsetQ)
        C=C+mcdlist
    return C

def createmcd1(Q,V,subgoalsQ):
    mcdlist = []
    subsetsV = chain.from_iterable(combinations(V.subgoals, r) for r
in range(1,(len(V.subgoals)+1)))
    for size, subsetV in enumerate(subsetsV):
        if viewExists(mcdlist,subsetV):
            continue
```

```
        mcd=createmcd2(Q,V,subgoalsQ,subsetV)
        if(mcd):
            mcdlist.append(mcd)
    return mcdlist

def createmcd2(Q,V,subgoalsQ,subsetV):
    if len(subsetV) > len(subgoalsQ):
        return
    subsetsV = chain.from_iterable(combinations(subsetV, r) for r in
range(1,(len(subgoalsQ)+1)))
    if not subsetsV:
        return
    for subgoalsV in subsetsV:
        mcd=createmcd3(Q,V,subgoalsQ,subgoalsV)
        if mcd:
            return mcd
    return

def createmcd3(Q,V,subgoalsQ,subgoalsV):
    #printSubsetSubGoal(subgoalsQ)
    #printSubsetSubGoal(subgoalsV)
    phi=defaultdict(str)
    hh=defaultdict(str)
    if len(subgoalsQ) != len(subgoalsV):
        return
    for i in range(len(subgoalsQ)):
        sgQ=subgoalsQ[i]
        sgV=subgoalsV[i]
        #print sgV.toString
        if (str(sgQ.name) != str(sgV.name)):
            return
        if (len(sgQ.args) != len(sgV.args)):
            return
        for i in range(len(sgQ.args)):
            argQ=sgQ.args[i]
            argV=sgV.args[i]
            if (not canMerge(argQ,argV,phi,hh,V.head.args)):
                return
            if (propertyMiniCon(Q,V,phi,hh,subgoalsQ)):
                mcd=MCD(Q, subgoalsQ, applyHHQuery(hh,V), subgoalsV,
phi, hh)
                return mcd
    return

def applyHHSubgoal(hh,sg):
    args=[]
    for a in sg.args:
        if a not in hh.keys():
```

```
            d=a
        else:
            d=hh[a]
        args.append(d);
    return SubGoal(sg.name, args);

def applyHHQuery(hh,Q):
    nsgs=[]
    nhead =  applyHHSubgoal(hh,Q.head)
    for sg in Q.subgoals:
        nsg=applyHHSubgoal(hh,sg)
        nsgs.append(nsg)
    return ConjQuery(nhead,nsgs)

def canMerge(argQ,argV,phi,hh,args):
    if not phi[argQ]:
        phi[argQ]=argV
        return True
    if phi[argQ]==argV:
        return True
    # argQ is mapped to a diff arg and not to argV
    if (argV not in args):
        if (phi[argQ] not in args):
            # one of them is not distinguised
            return False
    if not hh[argV]:
        hh[argV]=phi[argQ]
    return True

def propertyMiniCon(Q,V,phi,hh,subgoalsQ):
    #print phi
    for varQ in phi.iterkeys():
        varV = phi[varQ]
        if varV in V.head.args:
            continue
        # Property1
        if varQ not in Q.head.args:
            return False

        # Property2
        for subgoal in subgoalsQ:
            if (varQ in subgoal.args) and (subgoal not in subgoalsQ):
                return False
    return True

def queryExists(C,subgoalsQ):
    for mcd in C:
        subgoalsMCD = mcd.mappedQuerySubgoals
```

```
        if Set(subgoalsMCD).issubset(Set(subgoalsQ)):
             return True
    return False

def viewExists(C,subgoalsV):
    for mcd in C:
        subgoalsMCD = mcd.viewSubGoals
        if Set(subgoalsMCD).issubset(Set(subgoalsV)):
            return True
    return False

# initialze Conj query Q
def initializeQV():
    q1=SubGoal("q1",['v1','v2','v3'])
    g1=SubGoal("g1",['v1','v2'])
    g2=SubGoal("g2",['v3','v4'])
    g3=SubGoal("g3",['v3'])

    subgoalsQ=[]
    subgoalsQ.append(g1)
    subgoalsQ.append(g2)
    subgoalsQ.append(g3)

    Q=ConjQuery(q1,subgoalsQ)

    # initialize Views V1,V2,V3
    subgoalsV1=[]
    subgoalsV1.append(g1)
    subgoalsV1.append(g2)

    V1=ConjQuery(q1,subgoalsV1)

    subgoalsV2=[]
    subgoalsV2.append(g1)
    subgoalsV2.append(g2)
    subgoalsV2.append(g3)

    V2=ConjQuery(q1,subgoalsV2)

    subgoalsV3=[]
    subgoalsV3.append(g3)

    V3=ConjQuery(q1,subgoalsQ)

    V=[V1,V2,V3]
    return Q,V
```

```
def studentSchema():
    s1=SubGoal("student",['S','P','Y'])
    q=SubGoal("q",['S','P','Y'])

    subgoals=[]
    subgoals.append(s1)

    Q=ConjQuery(q,subgoals)

    s2=SubGoal("student",['S1','P1','Y1'])
    s3=SubGoal("taught",['P1','D1'])
    s4=SubGoal("program",['P1','C1'])
    v1=SubGoal("v1",['S1','Y1','D1'])
    v2=SubGoal("v2",['S1','P1'])
    v3=SubGoal("v3",['P1','C1'])

    subgoals=[]
    subgoals.append(s2)
    subgoals.append(s3)

    V1=ConjQuery(v1,subgoals)

    subgoals=[]
    subgoals.append(s2)

    V2=ConjQuery(v2,subgoals)

    subgoals=[]
    subgoals.append(s4)

    V3=ConjQuery(v3,subgoals)

    V=[V1,V2,V3]

    return Q,V

def citingPapersSchema():

    s1=SubGoal("cites",['x','y'])
    s2=SubGoal("cites",['y','x'])
    s3=SubGoal("sameTopic",['x','y'])
    q=SubGoal("Q1",['x'])

    subgoals=[]
    subgoals.append(s1)
    subgoals.append(s2)
    subgoals.append(s3)
```

```
    Q=ConjQuery(q,subgoals)

    s4=SubGoal("cites",['a','b'])
    s5=SubGoal("cites",['b','a'])
    v4=SubGoal("V4",['a'])

    subgoals=[]
    subgoals.append(s4)
    subgoals.append(s5)

    V4=ConjQuery(v4,subgoals)

    s6=SubGoal("sameTopic",['c','d'])
    v5=SubGoal("V5",['c','d'])

    subgoals=[]
    subgoals.append(s6)

    V5=ConjQuery(v5,subgoals)

    s7=SubGoal("cites",['f','g'])
    s8=SubGoal("cites",['g','h'])
    s9=SubGoal("sameTopic",['f','g'])
    v6=SubGoal("V6",['f','h'])

    subgoals=[]
    subgoals.append(s7)
    subgoals.append(s8)
    subgoals.append(s9)

    V6=ConjQuery(v6,subgoals)

    V=[V4,V5,V6]

    return Q,V

# initialze Conj query Q
def initializeQV_2():
    s1=SubGoal("student",['S','P','Y'])
    s2=SubGoal("taught",['P','D'])
    s3=SubGoal("program",['P','C'])
    q=SubGoal("q",['S','P','Y'])

    Q=ConjQuery(q1,[(SubGoal("student",['S','P','Y']))])

    # initialize Views V1,V2,V3
    subgoalsV1=[]
    subgoalsV1.append(s1)
```

```
    subgoalsV1.append(s2)

    V1=ConjQuery(q1,subgoalsV1)

    subgoalsV2=[]
    subgoalsV2.append(g1)
    subgoalsV2.append(g2)
    subgoalsV2.append(g3)

    V2=ConjQuery(q1,subgoalsV2)

    subgoalsV3=[]
    subgoalsV3.append(g3)

    V3=ConjQuery(q1,subgoalsQ)

    V=[V1,V2,V3]
    return Q,V

main()
```