Assignment - 1

Waghe Shubham Yatindra | 13MF3IM17

File Organization and Database Systems (MA60050)

```
# This PYTHON program finds the candidate keys for given set of FDs
'''input
ABCDEFGHIJ
A->G
HB->A
EA->C
JD->E
E->D
IFE->H
q
'''
import re
import itertools

fd_pattern = re.compile('^\s*[a-zA-Z]+\s*->\s*[a-zA-Z]+\s*$')

class FD:
    def __init__(self, left, right):
        self.left = left
        self.right = right

    def __repr__(self):
        return self.left + '->' + self.right

    def get_left(self):
        return self.left

    def get_right(self):
        return self.right

def check_valid_fd(fd):
    fd = fd.replace(" ","")
    prob_attrs_check = list(fd.replace("->",""))
    for attr in prob_attrs_check:
        if attr not in all_elems:
            print "ERROR: Unknown attribute(s) used, try again"
            return False
    if not fd_pattern.match(fd):
        print "ERROR:: Badly formatted FD - must be of the form AC->B, try again"
        return False
    else:
        current_fd = fd.split('->')
        return FD(current_fd[0].strip(), current_fd[1].strip())

def generate_closure(attrs, fd_set):
    closure = attrs
    found = True
    while(found):
        found = False
        for fd in fd_set:
            if set(fd.get_left()).issubset(set(closure)):
                for c in fd.get_right():
                    if not c in closure:
                        closure = closure + c
                        found = True

    return closure
```

```
def get_all_combinations(s):
    length = len(s)
    all_combinations = []

    for i in range(1, len(s)+1):
        all_combinations = all_combinations + [ ''.join(chars) for chars in itertools.combinations(s, i) ]

    return all_combinations

# Subset Check
def check_subset(string, CANDIDATE_KEYS):
    for key in CANDIDATE_KEYS:
        p, q = sorted(key), sorted(string)
        if set(p).issubset(q):
            return True
    return False

if __name__ == '__main__':
    print '*****************************************************************'
    print 'This program finds the candidate keys for given FDs'
    print '*****************************************************************'
    print 'Enter all the attributes'

    # all_elems = list(set(raw_input().strip().split()[0]))
    all_elems = list(raw_input().strip().replace(",","").replace(" ","").split()[0])
    attr_count = len(all_elems)

    print 'All entered elements: ' + ','.join(all_elems)
    print '*****************************************************************'

    all_fds = []
    FD_collection = []
    while True:
        input = raw_input('Enter FD in the format AB->C, type Q to exit: ').strip()
        if input == 'q' or input == 'Q':
            break
        else:
            v = check_valid_fd(input)
            if v != False:
                FD_collection.append(v)
    print FD_collection
    LEFT = []
    RIGHT = []
    MIDDLE = []
    # print all_fds

    fd_string = ','.join(all_fds)
    # print fd_string

    for fd in FD_collection:
        LEFT += list(fd.get_left())
        RIGHT += list(fd.get_right())

    LEFT = list(set(LEFT))
    RIGHT = list(set(RIGHT))
    LEFT_ONLY = [left for left in LEFT if left not in RIGHT]
    RIGHT_ONLY = [right for right in RIGHT if right not in LEFT]

    # print LEFT_ONLY
    # print RIGHT_ONLY
    SIDE = LEFT_ONLY + RIGHT_ONLY
    MIDDLE = [middle for middle in all_elems if middle not in SIDE]
    # print MIDDLE
```

```
MIDDLE_COMBINATIONS = get_all_combinations(''.join(MIDDLE))
middle_count = len(MIDDLE_COMBINATIONS)
# print middle_count

LEFT_STRING = ''.join(LEFT_ONLY)
string = LEFT_STRING
i = 0
m = ''

CANDIDATE_KEYS = []
key_found = False
key_length = attr_count
while True:
    string = LEFT_STRING + m
    if len(string) <= key_length or not key_found:
        closure = generate_closure(string, FD_collection)
        if len(list(closure)) == attr_count:
            if not check_subset(string, CANDIDATE_KEYS):
                CANDIDATE_KEYS.append(string)
            key_length = len(string)
        if i < middle_count:
            m = MIDDLE_COMBINATIONS[i]
            i += 1
        elif i >= middle_count:
            break
    else:
        break

print '*****************************************************************'
print 'Candidate Key(s): ' + ','.join(CANDIDATE_KEYS)
print '*****************************************************************'
```

OUTPUT :

```
*****************************************************************
This program finds the candidate keys for given FDs
*****************************************************************
Enter all the attributes
ABCDEFGHIJ
All entered elements: A,B,C,D,E,F,G,H,I,J
*****************************************************************
Enter FD in the format A->C, type q to exit: A->G
Enter FD in the format A->C, type q to exit: HB->A
Enter FD in the format A->C, type q to exit: EA->C
Enter FD in the format A->C, type q to exit: JD->E
Enter FD in the format A->C, type q to exit: E->D
Enter FD in the format A->C, type q to exit: IFE->H
Enter FD in the format A->C, type q to exit: q
*****************************************************************
Candidate Key(s): BFIJD,BFIJE
*****************************************************************
```