(1) How would you figure out who has the most connections in this network?

This query will return all the persons having most connections in the network.

```
select person from (select person , dense_rank() over (order by connections desc) as rnk
from (select a.person_1 as person, count(a.person_2) as connections from social_network
a  group by a.person_1) ) where rnk = 1      ;
```

If we want any one of the person having most connections than we can use following query.(this is oracle specific syntax , for postgres we can use limit)

```
select person from (select a.person_1 as person, count(a.person_2) as connections from
social_network a
group by a.person_1 order by connections desc ) where rownum = 1     ;
```

(2) How would you figure out who has the most connections in common with an arbitrary member of the
network? (e.g., Malia has 4 connections [her parents, the two younger Bushes])

This query will return all the person having most common connections with any other member , amongst all possible member-pair.
In this also if we want only one person then we can use rownum and order by in oracle or limit in postgres.

```
select substr(member_pair , 0,instr(member_pair , '-',1) - 1) as Person from
(select member_pair , dense_rank() over (order by mutual_friends desc) as rnk from
(select a.person_1 || '-' || b.person_1 as Member_Pair , count(*)  as mutual_friends  from
social_network a , social_network b where a.person_1 <> b.person_1 and a.person_2 =
b.person_2 group by a.person_1 || '-' || b.person_1 )) where rnk = 1   ;
```

(3) How would you figure out the number of common connections between any/all pair of connected members (e.g., Barack and George have 2 common connections [Michelle and Laura]).

```
select a.person_1 || '-' || b.person_1 as Member_Pair , count(*)  as mutual_friends  from
social_network a , social_network b where a.person_1 <> b.person_1
and a.person_2 = b.person_2  group by a.person_1 || '-' || b.person_1 order by   member_pair  ;
```

Above query will display all the possible combination (e.g the pairs obama-sasha , sasha-obama will be treated different) to display only the unique pairs we can use the below query. The query will work when (a.user_id * b.user_id) is unique for every pair , it will always work if user_id's are prime numbers.

```
select person|| '-' || friend as Member_Pair , count(*)  as mutual_connections from
(select a.person_1 as Person , b.person_1 as Friend , b.person_2 as Mutual_Friend ,
row_number() over (partition by (a.user_id * b.user_id) , a.person_2 order by a.user_id ) as ord
from social_network a , social_network b where a.person_1 <> b.person_1
and a.person_2 = b.person_2) x where ord = 1  group by person||'-' || friend order by
mutual_connections desc ;
```

(4) Now imagine our network is much bigger than 8 people, say 10000 people. Now we're potentially into the tens of millions of connections. Is SQL still a good choice for pulling this information on the number of common connections between any/all pairs of connected members. Or would you suggest some other tool?

If we have potentially many connections then SQL is not a good choice since in SQL we will have to perform self join to get the information , which will potentially compare every row with every other row stored in the database. Also in SQL we don't a way to cache data we have already seen to avoid redundant calculations. For this specific use case we can batch process the data using map reduce or we can store the data in key-value data store which supports efficient set operations. This will be faster than SQL.

(5) If you'd suggest moving beyond SQL, how might you approach the problem?

Approach1 : Using Mapreduce :
We can export the data stored in sql table to a .txt file or sequence file in hdfs or we can read data from database.

Mapreduce 1 :
This job will output the list of friends with key as person and value as list of friends.

```
def mapper() :
   for line in .txt file :
       person , friend = line.split()
       yeild person , friend

def reducer(key , value) :  #value will be list of friends for every key , this will be generated as part
of sort and shuffle phase.

    output_file.write(key , value)
```

Mapreduce 2 :
The input to this job will output file obtained from Mapreduce1

```
def mapper (self , file)
   for line in file :
        l = line.split()
        person , friend_list = l[0] , l [1:]
        for line2 in file :
             l1 = line.split()
             person1 , friend_list1 = l1[0] , l1[1:]
             yield (person , person1) , (friend_list , friend_list1)

 output of the map will be like this :
 (A - B) , (A C D E) (B C D)
 (A-C) , (A B D E)(B C D)

def reducer (self , key , value) :
      output_value = value[0].intersect(value[1])

      return key , output_value

output from reducer will be :
(A-B) , (C , D)
(A-C) , (B, D)
'''
```

Approach2 : Using Redis

we can use redis to store connections of person as a key : value , where key is person's name and value will be set of person's friends.
we can then iterate over person's and use redis's sinter (set intersection) to find common connections.
This way we will be iterating over the person's only one time as compared to sql where we were doing cross join.
Redis is also fast since all the data stored is in memory as compared to sql database where there will be a lot of I/o’s.

```
class MutualFriends(object):

    def __init__(self , host , port):
 # creating connection to redis db , for simiplicity i have not added error handling.
            self.connection = redis.StrictRedis(
                    host=host,
                    port=port,
                    db=0
                )
    def mutual_friends(self, person1 , person2):
        mutual_connections = self.connection.sinter(person1,person2)

        return len(mutual_connections)

    def friend_list(self, key):
         return self.connection.smembers(key)

    def number_of_common_connections(self):
        common_connections = {}
# if keys more than we can iterate the keys in batches , however here I am iterating them one by
one
        for person in self.connection.scan_iter("person:*") :
            for friend in self.connection.scan_iter("person:*"):
                if person == friend :
                    continue
                if friend+'-'+person in  common_connections  :
                    continue
                common_connections[person+'-'+friend] = len(self.mutual_friends())

        return common_connections


if __name__ == '__main__':
    host = 'localhost'
    port = '6667'

    a = MutualFriends(host,port)

    list_connection_pair = a.number_of_common_connections()
```

(6) Use Github's open API to return all members of the following repo: https://github.com/vinay-shipt/detake-
home (any language you want to use works for this, but Python or Ruby would be preferred)

```
'''
Here I have used requests library to make a call to github's api , however we can also use curl and
get the response from the api.

In the question 6 , it asks for members of repo , however I could not find documention of github api
for members of repo. Members usually belong to
oraganisation not repo. Repo has contributer and collaborators.

I was getting error while find collaborators since I dont have push access to the given repo.
However the code will be similiar. This below code is
implementation to find number of contributers.
'''
import requests
import json
def list_members(username,password,repo_url) :
    api_base_url = 'https://api.github.com'
    repo_attributes = repo_url.split('/')
    owner = repo_attributes[3]
    repository = repo_attributes[4]
    if owner == '' or repository == '':
        print('enter valid repository url')
        return
    #r = requests.get('{2}/repos/{0}/{1}/collaborators'.format(owner,repository,api_base_url),
auth=(username, password))
    r = requests.get('{0}/repos/{1}/{2}/
contributors'.format(api_base_url,owner,repository),auth=(username, password))

    if r.status_code == 200:
       data =  json.loads(r.content.decode('utf-8'))
       print(data)
       member_list = []
       for i in data :
          if i['login'] :
             member_list.append(i['login'])
       return member_list
    else:
       print(r.status_code)
       return None
# parameter for api authentication
api_username = 'xxx'
api_password = 'xxx'
repository_url = 'https://github.com/vinay-shipt/de-take-home'
members = list_members(api_username,api_password,repository_url)
if members :
   print(members)
else :
   print('No members for the specified repository')
```