SQL questions:

Given "**Employee**" table below, please write the following ***SQL statements*** :

Employee

id	Name	salary	manager_id
1	John	300	3
2	Mike	200	3
3	Sally	550	4
4	Jane	500	7
5	Joe	600	7
6	Dan	600	3
7	Phil	550	NULL
…	…	…	…

1. Give the names of employees, whose salaries are greater than their immediate managers':

```
select distinct Name from Employee A Join Employee B on A.manager_id = B.id WHERE A.salary > B.salary
```

2. What is the average salary of employees who do not manage anyone? In the sample above, that would be John, Mike, Joe and Dan, since they do not have anyone reporting to them.

```
select avg（salary） from Employee where id not in （select distinct manager_id from employee where manager_id != NULL）
```

q/KDB+ questions:

Employee query.

Repeat the previous SQL query but write the KDB equivalent query

1.

```
select Name from (Employee lj ([manager_id:Employee`id] wage:Employee`salary)) where (not null manager_id) and (salary>wage)
```

2.

```
select avg salary from Employee where (not null manager_id) and (not id in Employee`manager_id)
```

Exists?

Write a function 'e' which takes a variable symbol v and returns 1b if v is defined and 0b if it is not:

```
     a:10
     exists`a
 1b
     exists`b
 0b
```

```
exists:{x in key `.}
```

Dictionary to list.

Write a function 'undict' which takes a nested dictionary and replaces each dictionary with a pair of lists: (symbols;values):

```
       d:`a`b!(`c`d`e!10 20 30;`f`g!(40;`h`i`j!50 60 70))
       undict d
     (`a`b;((`c`d`e;10 20 30);(`f`g;(40;(`h`i`j;50 60 70)))))
```

```
undict : {:$[99h=type x;(key x;.z.s each value x);x]}
```

Infinite loop.

Without using a loop or recursion, find an expression which causes an infinite loop.

```
{sqrt x}\[{x>0};100]
```

Depends on me.

In q we define a view or "dependency" with '::', e.g.:

```
     a:10
     b::a+1
     c::a+2
     d:c+20
     e::b+d
     f::e+4
     g::f+5
```

In this example, if a is reassigned, then b, c, e, f and g are invalidated. Referencing an invalid variable causes its definition to re-compute and return a new value. In general if any variable is invalidated, then all of its descendants are invalidated.

The primitive .z.b takes one or more symbols s and for each symbol k in s returns a vector of symbols of variables which *directly* depend on k:

```
  q) .z.b`a`d
  `b`c
  ,`e
```

Write a function 'dependson' which takes a single symbol v and returns a list of ALL the variables which are invalidated by assignment to v, e.g.:

```
  q) dependson`a
  `a`b`c`e`f`g
```

```
dependson:{if[doesExist x; .z.s each .z.b 0N! x;]}
```

Who moved?

You're given a vector of unique elements:

```
  v:10 20 30 40 50 60 70
```

Some operation has moved a single one of the elements into a new position:

```
  w:10 20 70 30 40 50 60   / 70 inserted between 10 and 20
```

There is a function which returns the index of the repositioned element:

```
  moved[v;w]
2
  moved[10 20 30;20 30 10]
2
  moved[10 20;20 10]
0
```

Write 'moved'.

```
v:10 20 30 40 50 60 70
w:10 20 70 30 40 50 60
moved :{first where ((y?x) - (til count y))=1}
```

Maximum Overlap Intervals.

Given the following table:

```
procs:([]id:10*1+til 8;anest:`baker`baker,6#`dow;start:"t"$08:00 09:00
09:00 08:00 10:00 12:30 13:30 18:00;end:"t"$11:00 13:00 15:30 13:30
11:30 13:30 14:30 19:00)
```

(i) Write a query to list out the ids each row is intersecting with. Append answer to the last column as w

(ii) Write a query to determine for each anest the max # intersecting ids over periods of intersection. Append answer as the last column as s

Expected outputs:

```
id anest start        end          s w
-----------------------------------------------
10 baker 08:00:00.000 11:00:00.000 2 10 20
20 baker 09:00:00.000 13:00:00.000 2 10 20
30 dow   09:00:00.000 15:30:00.000 3 30 40 50 60 70
40 dow   08:00:00.000 13:30:00.000 3 30 40 50 60
50 dow   10:00:00.000 11:30:00.000 3 30 40 50
60 dow   12:30:00.000 13:30:00.000 3 30 40 60
70 dow   13:30:00.000 14:30:00.000 2 30 70
80 dow   18:00:00.000 19:00:00.000 1 ,80
```

```
(i)
len: count procs;
procs: update
head:start>=(len,len)#end,
tail:end<=(len,len)#start,
class:anest=(len,len)#anest from procs;
procs: update w:(class * (head+tail)=0)=1 from procs;
procs: update w:id @[w;::;where] from procs;
procs: select id,anest,start,end,w from procs;

(ii)
anestList: distinct procs`anest;
sList: ();
iAnest: 0;
while[iAnest<count anestList;
    target: anestList[iAnest];
    tb: select from procs where anest=target;
    timeList: tb`start;
    timeList,: tb`end;
    len: -1 + count timeList;
    tb: update head: start<=((count tb),len)#(timeList til len) from tb;
    tb: update tail: end>=((count tb),len)#(1+timeList til len) from tb;
    tb: update contained: head*tail from tb;
    tb: update overlap: contained*((count tb),len)#(sum contained) from
tb;
    tb: update s: @[overlap;::;max] from tb;
    sList,: tb`s;
    iAnest: iAnest+1];
procs: update s:sList from procs;
```

Average price computation.

Write a query to compute avgprc for trades in the attached file avgprc.csv.

The correct answer is the avgprc col of this file.
The format for avgprc.csv is:

```
     ("SFFF";enlist",")0:`:avgprc
```

The spec of avgprc is:

avgprc starts with the first prc where tran=`open.

it is calculated thus: when you are increasing your position (long or short) the formula is:

((avgprc*abs prev accumulated)+prc*abs trd)%abs accumulated

it does not change until you switch sides; i.e accumulated trd switches sign at which point it resets to prc.

Pascal Triangle

Create a function to compute N layer of pascal triangle.

```
Example for 10/11? layers:
q)pascal[10]
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

pascal:{{prior[+]x,0}\[x-1;1]}
```

R questions:

Maximum Overlap Intervals.

Repeat the q/KDB+ question in previous section and implement in R language instead. Table is given below as R's data.frame. Allow to use data.table, dplyr, xts, zoo package(s) or any preferred libraries at your discretion to complete the task.

```
R>
data.frame(id=seq(10,80,by=10),anest=c("baker","baker",rep("dow",6)),
start=c("08:00","09:00","09:00","08:00","10:00","12:30","13:30","18:00
"),end=c("11:00","13:00","15:30","13:30","11:30","13:30","14:30","19:0
0"))

  id anest start   end
1 10 baker 08:00 11:00
2 20 baker 09:00 13:00
3 30   dow 09:00 15:30
4 40   dow 08:00 13:30
5 50   dow 10:00 11:30
6 60   dow 12:30 13:30
7 70   dow 13:30 14:30
8 80   dow 18:00 19:00
```

	id	anest	start	end	s	w
1	10	baker	08:00	11:00	7	10
2	20	baker	09:00	13:00	8	10, 20
3	30	dow	09:00	15:30	20	30, 40
4	40	dow	08:00	13:30	19	40
5	50	dow	10:00	11:30	21	30, 40, 50
6	60	dow	12:30	13:30	21	30, 40, 50, 60
7	70	dow	13:30	14:30	21	30, 40, 50, 60, 70
8	80	dow	18:00	19:00	21	30, 40, 50, 60, 70, 80

Pascal Triangle

Repeat the q/KDB+ question in previous section and implement in R language instead.

```
> pascalTriangle(10)
[[1]]
[1] 1

[[2]]
[1] 1 1

[[3]]
[1] 1 2 1

[[4]]
[1] 1 3 3 1

[[5]]
[1] 1 4 6 4 1

[[6]]
[1]  1  5 10 10  5  1

[[7]]
[1]  1  6 15 20 15  6  1

[[8]]
[1]  1  7 21 35 35 21  7  1

[[9]]
[1]  1  8 28 56 70 56 28  8  1

[[10]]
 [1]   1   9  36  84 126 126  84  36   9   1

[[11]]
 [1]   1  10  45 120 210 252 210 120  45  10   1
```

Portfolio VaR & CVaR

Assume you have the following portfolio as of 2016/01/01:

AAPL.O	15%
IBM.N	20%
GOOG.O	20%
BP.N	15%
XOM.N	10%
COST.O	15%
GS.N	05%

i. Using historical daily returns (Yahoo/Google Finance or any other market data source), calculate VaR95% and CVaR95% of the portfolio as of 2016/12/31

```
> VaR(portfolio.r,p=0.95,method="historical")
    portfolio return
VaR      -0.01416877
> ETL(portfolio.r,p=0.95,method="historical")
   portfolio return
ES      -0.02125608
```

ii. Using expected mean, covariance matrix and parametric method, calculate VaR95% and CVaR95%.

```
> VaR(portfolio.r,p=0.95,method="gaussian")
    portfolio return
VaR      -0.01427473
> ETL(portfolio.r,p=0.95,method="gaussian")
   portfolio return
ES      -0.01805313
```

iii. Assume you can change weights, allow shorting but no leverage (i.e. sum of weights equal to 100%), and rebalance monthly. What is the optimal portfolio holding by end of each month, till end of 2016

```
$`2016-11-30`
***********************************
PortfolioAnalytics Optimization
***********************************

Call:
optimize.portfolio(R = R[1:ep, ], portfolio = portfolio, optimize_method = optimize_method,
    search_size = search_size, trace = trace, parallel = FALSE,
    rp = rp)

Optimal Weights:
   AAPL     IBM    GOOG      BP     XOM    COST      GS
 0.0412  0.2012  0.1668 -0.0789  0.3207  0.3578 -0.0088

Objective Measure:
  StdDev
0.008042



$`2016-12-30`
***********************************
PortfolioAnalytics Optimization
***********************************

Call:
optimize.portfolio(R = R[1:ep, ], portfolio = portfolio, optimize_method = optimize_method,
    search_size = search_size, trace = trace, parallel = FALSE,
    rp = rp)

Optimal Weights:
   AAPL     IBM    GOOG      BP     XOM    COST      GS
 0.0440  0.1983  0.1632 -0.0715  0.3188  0.3534 -0.0061

Objective Measure:
  StdDev
0.007917
```

- Notes: Please state your assumption(s), if any

Position calculator

Refer to the data file, “pos.csv”, “trd.csv”:

i. From “pos.csv”, calculate the netted position per each user

	user	net_pos
1	A	-6393
2	B	-1889
3	C	5430
4	D	-1379
5	E	2949

ii. List out all the boxed positions.
Boxed positions are defined as:
A trader has long (quantity > 0) and short (quantity < 0) positions for the same symbol at different brokers (pb)

Filter

	box_A_list	box_B_list	box_C_list	box_D_list	box_E_list
1	1183.T	1391.T	1042.T	1303.T	1436.T
2	1250.T	1723.T	1315.T	NA	NA
3	1492.T	1736.T	1368.T	NA	NA
4	1706.T	1786.T	1436.T	NA	NA
5	1775.T	NA	NA	NA	NA
6	1811.T	NA	NA	NA	NA

iii. From the “trd.csv”, assume all the orders arrived at the same time, find all the potential crossing. Create a file with the original quantity (qty), journal (jrnl) and trades (trd)columns

e.g

sym	user	qty	jrnl	trd
1310.T	A	-146	146	0
1310.T	B	1990	43	1947
1310.T	C	1889	41	1848
1003.T	A	816	0	816
1003.T	C	2411	0	2411
1003.T	D	2880	0	2880

(the “?” is where you need to calculate and fill out)

iv. From output in (iii.), find the total quantity to trade, group by sym

	sym	Total_trd
1	1003.T	6278
2	1020.T	5744
3	1022.T	-3312
4	1042.T	4688
5	1074.T	-4184
6	1084.T	7140
7	1093.T	-6403
8	1117.T	-2198
9	1172.T	5816
10	1183.T	5842
11	1234.T	-423
12	1249.T	1547
13	1250.T	1861
14	1275.T	-2745
15	1303.T	8988
16	1310.T	6561
17	1315.T	4327
18	1361.T	6568
19	1368.T	-2675
20	1391.T	-2015
21	1436.T	-11697
22	1491.T	-6028
23	1492.T	-4047
24	1497.T	-2474
25	1525.T	4814
26	1527.T	2417
27	1541.T	5142

Showing 1 to 27 of 48 entries

v. Using “pos.csv” and “trd.csv”, find the final position, per user, per sym. (assume all trades in trd.csv got fully executed; and the boxed positions all collapsed to the largest holding account)

	sym	user	Total_pos	Total_trd	final_pos
1	1003.T	A	0	816.000000	816.000000
2	1003.T	B	-14	0.000000	-14.000000
3	1003.T	C	-31	2411.000000	2380.000000
4	1003.T	D	0	2880.000000	2880.000000
5	1003.T	E	0	171.000000	171.000000
6	1020.T	A	0	1958.752936	1958.752936
7	1020.T	B	1292	1223.393967	2515.393967
8	1020.T	C	0	0.000000	0.000000
9	1020.T	D	0	0.000000	0.000000
10	1020.T	E	0	2561.853097	2561.853097
11	1022.T	A	0	-1174.404917	-1174.404917
12	1022.T	B	628	-1228.480983	-600.480983
13	1022.T	C	0	-909.114099	-909.114099
14	1022.T	D	98	0.000000	98.000000
15	1022.T	E	0	0.000000	0.000000
16	1042.T	A	0	908.274979	908.274979
17	1042.T	B	0	0.000000	0.000000
18	1042.T	C	515	2672.780529	3187.780529
19	1042.T	D	233	0.000000	233.000000
20	1042.T	E	-809	1106.944492	297.944492
21	1074.T	A	-724	-2449.796650	-3173.796650
22	1074.T	B	0	0.000000	0.000000
23	1074.T	C	0	-104.624676	-104.624676
24	1074.T	D	466	-1444.017929	-978.017929
25	1074.T	E	-239	-185.560745	-424.560745
26	1084.T	A	-608	0.000000	-608.000000
27	1084.T	B	0	805.318037	805.318037

Showing 1 to 27 of 238 entries

vi. Create a Unit test to check your calculation for the above questions. Think about what conditions should be check and passed or else should raise errors, etc.

Q6 The question description is very vague, and we could check the solution quality from serveral perspective: First is the data quality, we should cross check our data from mutli-sources. Second is the defination, we should check the definition of journal and trades, and compare our theoretical result with the market data. Third, we shoul loose our assumption and make our analysis close to the practice. For example, we cannot assume that all the orders arrive at the same time.