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 t1.Name
from Employee t1
left join Employee t2 on t1.manager_id=t2.id
where t1.salary > t2.salary
```

```
sh-4.3$ sqlite3 database.sdb < main.sql
Sally
Joe
Dan
sh-4.3$
```

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(t1.salary)
from Employee t1
where t1.id not in (select manager_id from Employee)
```

Answer: 425

q/KDB+ questions:

Employee query.
Repeat the previous SQL query but write the KDB equivalent query

Exists?
Write a function 'exists' 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
```

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)))))
```

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

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
```

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'.

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
```

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 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
```

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	8:00	11:00	2	10,20
2	20	baker	9:00	13:00	2	10,20
3	30	dow	9:00	15:30	3	30,40,50,60,70
4	40	dow	8:00	13:30	3	30,40,50,60
5	50	dow	10:00	11:30	3	30,40,50
6	60	dow	12:30	13:30	3	30,40,60
7	70	dow	13:30	14:30	2	30,70
8	80	dow	18:00	19:00	1	80

Pascal Triangle

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

```
x=1
for (i in 1:10) {x=c(0,x)+c(x,0); print(x)}
```

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%

Data is from 2016/01/01 to 2016/12/31

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

First calculate the daily VaR from historical returns, and convert to annual VaR. Same with CVaR

VaR1
0.3006908
CVaR1
0.4641569

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

First calculate the mean and standard deviation from historical daily return and convert to annual. Then use $\mu - \sigma Z_{0.95}$ to calculate the VaR and CVaR.

VaR2
0.1483212
CVaR2
0.2327403

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

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

Solution: use dynamic optimization to rebalance the portfolio monthly. Our objective function is to minimize the **VaR95%** at 2016/12/31.

Assumption: for each of the stocks, we use historical return from 2004/08/20 to 2015/12/31 to calculate its daily mean and standard deviation as well as the covariance matrix among different stocks, and convert to monthly return($\boldsymbol{\mu}$), standard deviation($\boldsymbol{\sigma}$), and covariance matrix $\boldsymbol{\Sigma}$. We assume the monthly returns $(\boldsymbol{r})$ are i.i.d. and normally distributed from 2016/01/01 to 2016/12/31. Suppose we start from $P_0 = \$1$ at 2016/01/01 and we will end up with P_{12} in 12 months.

Objective function: Minimize $VaR95\%(P_{12} - P_0)$ = Maximize $Z_{95\%}(P_{12})$

Where μ_{12}, σ_{12} are determined each month by the weight vector $\boldsymbol{w_k}$. For example, at the beginning of month 12, we have P_{11} dollars in our account, and we decide the weight w_{12} to maximize
$Z_{95\%}(P_{12}) = Z_{95\%}(P_{11}(r^T w_{12} + 1)) =$
$P_{11}\,(1 + \mu^T w_{12}) - P_{11} Z_{0.95}\sqrt{(1 + 2\mu^T\, w_{12} + w_{12}^T \Sigma w_{12}) - [\,(1 + \mu^T w_{12})]^2}$. Note that P_{11} is the information we already observe at the beginning of month 12. Also, in this problem, the optimal decision w_{12}^* is independent of P_{11}.

Now moving back 1 month to the beginning of month 11 and we observe the portfolio value to be P_{10}. We need to maximize

$$
\begin{aligned}
&Z_{95\%}[P_{10}(1 + r^T w_{11})\,(1 + r^T w_{12})] \\
&= E[P_{10}(1 + r^T w_{11})\,(1 + r^T w_{12})] - Z_{0.95}\sqrt{var([P_{10}(1 + r^T w_{11})(1 + r^T w_{12})]\,)} \\
&= P_{10}(1 + \mu^T w_{11})\,(1 + \mu^T w_{12}) \\
&- P_{10} Z_{0.95}\sqrt{(1 + 2\mu^T\, w_{11} + w_{11}^T \Sigma w_{11})(1 + 2\mu^T\, w_{12} + w_{12}^T \Sigma w_{12}) - [(1 + \mu^T w_{11})\,(1 + \mu^T w_{12})]^2}
\end{aligned}
$$

Note the decision variable is only w_{11} since we already know w_{12}.

With this procedure, we can iterate backwards from month 12 all the way to month 1, and find all the optimal weights accordingly. Results are shown below:

Month	AAPL	BP	COST	GOOG	GS	IBM	XOM	optimal value
1	32%	5%	26%	9%	0%	9%	19%	0.87
2	32%	-15%	42%	23%	-12%	11%	18%	0.87
3	30%	-15%	42%	23%	-12%	12%	20%	0.87
4	29%	-18%	35%	26%	-11%	18%	21%	0.87
5	29%	-14%	42%	22%	-11%	13%	18%	0.87
6	28%	-13%	43%	21%	-11%	13%	19%	0.87
7	27%	-13%	41%	21%	-11%	14%	20%	0.88
8	30%	3%	35%	4%	1%	4%	23%	0.88
9	21%	-2%	37%	17%	-10%	20%	17%	0.89

10	21%	-3%	46%	19%	-8%	15%	11%	0.90
11	19%	-6%	40%	17%	-11%	23%	18%	0.91
12	16%	-4%	39%	15%	-11%	26%	19%	0.93

Position calculator

Refer to the data file, "pos.csv", "trd.csv":

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

Group by each user and symbol pair, and sum all positions.

See "netPositions.csv" in the output folder

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)

Group by user and symbol, and find in each group that has a long position in one pb and short position in another.

See "boxPositions.csv" in the output folder

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	?	?
1310.T	C	1889	?	?
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)

Note that, in this problem, we assume journal quantity is proportional to the quantity submitted, and partial quantity is possible. However, in reality it is not. So we can use a rounding scheme to find out the integer number of trades for each user.

See "journal.csv" in the output folder without rounding
See "journal2.csv" in the output folder with rounding

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

Group by symbol and sum the trd from previous table.
See "toTrade.csv" in the output folder

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)

Group by user and symbol, sum all the positions to collapse boxed positions. Then join this table with trd.csv table.

See "finalPositions.csv" in the output folder

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.

Generally, check if any NA values or NULL values or infinity in user or sym, otherwise raise error.

- For problem 1, check if there is any NA values or infinity in the position.
- For problem 2, check if there are more than 1 positions (with different signs) for the same user, same symbol and same broker.
- For problem 3 and 4, if we use rounding scheme, there is a problem: theoretically there could be a case where a user gets more than he wants because of the rounding. For example, if each buy user wants to get one 1 quantity, but there are 2 selling orders in total, then using the rounding scheme before, one user is going to get 2 quantity and other users are going to get 0 quantity. We should check for this case and avoid it by removing the additional quantity to other users.
- For problem 5, check if there are NA values in pos or qty.