MILLENNIUM INTERVIEW TEST SOLUTIONS

Jing Long Hoelscher

MARCH 9, 2017

Contents

SQL questions:

Employee query

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 a.Name as Name
from Employee a
left join
(select id, salary from Employee) m
on a.manager_id = m.id
where a.salary > m.salary
and a.manager_id is not NULL
```

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) as SalaryAverage
from Employee
where id not in
(select manager_id from Employee)
```

q/KDB+ questions:

Employee query

Repeat the previous SQL query but write the KDB equivalent query

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'.

```
Employee:("SSFS";enlist",")0:`$"C:\\Users\\debbi\\Documents\\Millenniu
m\\Employee.csv"

select Name from (Employee lj (1! (select manager_id:id,
managerSalary:salary  from Employee))) where salary >managerSalary
```

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 not id in exec manager_id from
Employee
```

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
```

```
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)))))
```

```
f:{$[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.

```
{(x+1)}\[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:{$[0<count(.z.b x);distinct raze (x;.z.b x;raze .z.s
each(.z.b x));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'.

```
moved:{:$[x[(x=y)?0b]=y[((x=y)?0b)+1];(x=y)?0b;-1+(count(x))- (reverse
x = y)?0b]}
```

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)

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

```
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)
tmp: procs lj select s: start, e: end, ids: id by anest from procs
interInvs: {[x1;y1;x2;y2] not ((y2 <= x1) | (y1<=x2))}
tmp: update isin: interInvs[start;end;s;e] from tmp
tmp: update w: {x where y}'[ids;isin] from tmp
procs: procs lj `id xkey select id,w from tmp
```

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
```

```
procs: `anest`start xasc procs
```

```
starts: select startNum: count i by anest, tm: start from procs
ends: select endNum: count i by anest, tm: end from procs
tmp2: `anest`tm xasc starts uj ends
tmp2: update numint: sums (0^startNum)-(0^endNum) by anest from tmp2
tmp3: procs lj select tms:tm,numints:numint by anest from tmp2
inInter: {[x;y;z] (x<=z) & (z<=y)}
tmp3: update isin: inInter[start;end;tms] from tmp3
tmp3: update s: {max(x) where y}'[numints;isin] from tmp3
procs: procs lj `id xkey select id,s from tmp3
procs: select id, anest, start,end, s, w 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.

There are three different ways to interpret the avgcost, and I am giving the avgcost values for all three methods.

- The first method is the requested one as in the test:
  ((avgprc*abs prev pos)+prc*abs trd)%abs accumulated

Where the denominator is defined as

$$\text{"}abs\ accumulated\text{"} = (abs\ prev\ pos) + abs\ trd$$

```
/* calculate avgcost as: ((avgcost * abs prev pos) + prc * abs trd) % ((abs prev pos)
+ abs trd) */
avgcost:("SFFF";enlist",")0:`$"C:\\Users\\debbi\\Documents\\Millennium\\avgcost.csv";
avgcost: update avgcost: ?[(tran=`open) | 0 >= (prev pos) * pos; prc; 0n] from
avgcost
CalcOne: {[tblin] update avgcost: (((prc*abs trd) + (prev avgcost)*(abs prev
pos))%((abs prev pos) + abs trd)) ^ avgcost from tblin}
avgcost: avgcost {[x;y] CalcOne[x]}/til (exec count tran from avgcost)
```

- The second approach to this problem uses the formula ((avgprc*abs prev accumulated)+prc*abs trd)%abs accumulated

where the denominator is defined as

$$\text{"}abs\ accumulated\text{"} = sums\ abs\ trd$$

```
/*alternative way (method2) to calculate average cost as: ((avgcost * abs prev
absCum) + prc * abs trd) % ((abs prev absCum) + abs trd) */
avgcost: update absCum: sums(abs trd) from avgcost
avgcost: update avgcost2: ?[(tran=`open) | 0 >= (prev pos) * pos; prc; 0n] from
avgcost
CalcOne2: {[tblin] update avgcost2: (((prc*abs trd) + (prev avgcost2)*(prev
absCum))%((prev absCum) + abs trd)) ^ avgcost2 from tblin}
avgcost: avgcost {[x;y] CalcOne2[x]}/til (exec count tran from avgcost)
```

- The third approach is to take the average cost as the total cost divided by the total position:

$$((avgprc * prev\ pos) + prc * trd)\%pos$$

```
/*alternative way (method3) to calculate average cost as: ((avgcost * prev pos) + prc
* trd) % pos */
avgcost: update avgcost3: ?[(tran=`open) | 0 >= (prev pos) * pos; prc; 0n] from
avgcost
CalcOne3: {[tblin] update avgcost3: (((prc*trd) + (prev avgcost3)*(prev pos))%pos) ^
avgcost3 from tblin}
avgcost: avgcost {[x;y] CalcOne3[x]}/til (exec count tran from avgcost)
```

Please see the results in the attached avgcost.csv file.

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

```
pascal:{{prior[+]x,0}\[x;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","19:00"))

```
  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
```

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

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

Code: please see solution_part1.R
Results: Please see the file procs.csv and below

```
> procs
   id anest start   end s              w
1: 10 baker 08:00 11:00 2          10 20
2: 20 baker 09:00 13:00 2          10 20
3: 30   dow 09:00 15:30 3 30 40 50 60 70
4: 40   dow 08:00 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.

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
```

Code: please see below and in the file solution_part1.R

```
pascal.triangle <- function(n){

  # Define the output to be a vector of list
  output <- vector('list',n)

  # Initilize the output vector
  output[[1]] <- 1

  # Display the output
  print(paste0("The Pascal Triangle for ", n, " is below:"))
  print(output[[1]])

  # Define recursively of the Pascal Triangle
  for(i in 2:(n+1)){
    output[[i]] <- c(output[[i-1]],0) + c(0,output[[i-1]])
    print(output[[i]])
  }
  return(output)
}
> pascal    <- pascal.triangle(10)
[1] "The Pascal Triangle for 10 is below:"
[1] 1
[1] 1 1
[1] 1 2 1
[1] 1 3 3 1
[1] 1 4 6 4 1
[1]  1  5 10 10  5  1
[1]  1  6 15 20 15  6  1
[1]  1  7 21 35 35 21  7  1
```

```
[1]  1  8 28 56 70 56 28  8  1
 [1]   1   9  36  84 126 126  84  36   9   1
 [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%

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

Code: please see solution_part1.R
Results: Please see below, also portfolioVaR.csv

```
> hvar
        5%
0.01418251
> hcvar
[1] 0.02031518
```

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

Assumptions

- The portfolio return is normally distributed and is serially independent.
- Historical data as the future distribution.
- Since the expected portfolio return is very close to zero, i.e. -0.0004957029, we assume the expected mean is zero. The formula to get the parametric VaR and CVaR is:

$$VaR = \sigma \cdot \Phi^{-1}(1-\alpha),$$
$$CVaR = \frac{\sigma}{\alpha} \cdot \phi\big(\Phi^{-1}(1-\alpha)\big),$$

Where α is the confidence level, and Φ is the normal cumulative distribution function, and ϕ is the normal density function.

Code: please see solution_part1.R
Results: Please see below, also portfolioVaR.csv

```
Results:
> parvar
           parvar
parvar 0.01495238
> parcvar
           parcvar
```

```
parcvar 0.01875088
```

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

We are going to make the following assumptions:

- The rebalance takes place at the end of each month.
- There is no transaction cost.
- For each risk-aversion λ, the solution to the zero-cost utility function

  $$u = h^T \cdot E(r) - \lambda \cdot h^T \cdot Cov \cdot h,$$
  $$subject\ to\ h^T \cdot e = 1, where\ e = (1,1,\cdots,1)^T,$$

  Is the solution:

  $$h_{opt} = \left(1 - \frac{r^T \cdot Cov^{-1} \cdot e}{2\lambda}\right) \cdot \frac{Cov^{-1} \cdot e}{e^T \cdot Cov^{-1} \cdot e} + \frac{Cov^{-1} \cdot r}{2\lambda},$$

  Which is a combination of the min-variance portfolio and a market portfolio. Here we will use the Covariance matrix based on the rolling window of past 250 days of data.

Code: please see solution_part1.R
Results: Please see the file optimalPortfolio.csv for the case the risk aversion parameter $\lambda = 0.5$

Position calculator

Refer to the data file, "pos.csv", "trd.csv":
From "pos.csv", calculate the netted position per each user

Code: please see solution_part2.R
Results: Please see below and the file nettedPos.csv

```
> pos.net
   user   pos
1:    A -6393
2:    B -1889
3:    C  5430
4:    D -1379
5:    E  2949
```

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)

Code: please see solution_part2.R
Results: Please see the file boxedPos.csv

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)

We assign the crossing according to the portion of the original position.
Code: please see solution_part2.R
Results: Please see the file trdCrossing.csv

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

Code: please see solution_part2.R
Results: Please see the file trdTotal.csv

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)

Code: please see solution_part2.R
Results: Please see the file pos.final.csv

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.

I will use testthat package to unit test the functions in solution_part2.R. The tests are in the script test_solution_part2.R. The script to run the unit tests are in run_tests.R. Please notice to place all three files solution_part2.R, test_solution_part2.R, and run_tests.R in the same directory with R readable path "path". In my case, everything is in my current folder, so my path is defined to be ".", please use your correct corresponding path for your folder which contains all the files before run the script run_test.R.

Code: please see test_solution_part2.R, test_solution_part2.R, and run_tests.R.
Results: Please see the test result in the file test_results.csv