AquaQ Training

Kdb+ Training Exercises : Answers

email:
support@aquaq.co.uk

web:
www.aquaq.co.uk

AquaQ Analytics

Contents

Installing Kdb+

Exercises from Notes: Answers

1. Download the trial version of the interpreter from:

```
http://www.kx.com/trialsoftware.php
```

2. The entire directory should be moved to (for windows):

```
C:\q\
```

3. Starting a process (for windows)

```
c:\q\w32\q.exe
```

4. Exits the q session

```
exit 0
```

Chapter 1

Atoms and Basic Operations

1. Start a new q session.

```
From the command prompt type "q"
```

2. Define the following variables:

```
q)/- assignment is made with : in q
q)a:5
q)b:a-3
q)c:1+3*b
```

3. Repeat question 2 in one single assignment statement.

```
q)/- right to left assignment, so we can do this:
q)c:1+3*b:-3+a:5
```

4. Calculate $\frac{a^2-3c}{b}$ and store the answer as the variable y. (Use as few brackets as possible)

```
q)/- Calculate a^2... and store as y.
q)y:((-3*c) + a*a)% b
q)
q)/- what is the type of y?
q)type y
-9h
q)/-y is a float
q)/- what is the result of typing y=b and y~b
q)y=b
1b
q)y~b
0b
q)/- = tests the value, ~ tests also the type hence it fails
q)/- How can you amend y so that y=b and y~b agree?
q)/- Cast y to be the same type as b
q)y:`long$y
q)y~b
1b
q)/-multiply b by 5 in place
q)/-use compound assignment
q)b*:5
```

5. Find the type values of the following objects:

```
q)type 123
-7h
q)type 123f
-9h
q)type 2014.09
-9h
q)type 2014.09m
-13h
q)type `me
-11h
q)type "you"
10h
q)
```

6. Which of the following objects have the same type? What are the types of the other objects?

```
q)/- Which of the following have the same type?
q)/- the 4.2... and 9.8 are both floats - the others:
q)type 4.29999871
-9h
q)type 7j
-7h /-long
q)type 4.2e
-8h /-real
q)type 0b
-1h /-boolean
q)type 9.8
-9h
q)
```

7. Get today's date and store it as the variable d.
   - Calculate the number of days since last Christmas.
   - Find out on which day of the week that the 10th of January 2011 was (hint: try using mod with a date)

```
q)/- Get today's date and store it as the var d
q)d:.z.d
q)
q) /-Days since last christmas - we can use algebra with dates
q)/-(ints underneath!)
q)d - 2013.12.25
105i
q)
q)/- Day of the week for 10th Jan 2011
q)2011.01.10 mod 7
2i
q)/- starts on a sunday, so 10th jan was a monday
q)/- (test with a date you know - today!)
q)d mod 7
4i
q)/- in the present, today is wednesday !
q)
```


8. Define the following items, and then cast as appropriate to the stated type:

```
q)
q)d1:"2014.01.01"
q)/-use "D" for strings
q)"D"$d1
2014.01.01
q)d2:`2013.12.10
q)/-requires casting first to a string
q)"D"$string d2
2013.12.10
q)n1:3.14
q)`int$n1
3i
q)/-note rounding for ints
q)n2:"2"
q)"I"$n2
2i
q)a1:"abcde"
q)`$a1
`abcde
q)a2:"abcde"
q)
q)"i"$a2
97 98 99 100 101i
q)
```

9. Show all of the variables defined in the current q session.

```
q)
q)\v
`a`a1`a2`b`c`d`d1`d2`n1`n2`y
q)
```

10. Close the current session.

```
q) \\
```


Chapter 2

Lists

1. Create two lists, a with values 1 2 3 and b with values 4 5.

```
q)a: 1 2 3
q)b: 4 5
q)
q)/- Join the lists to create c
q)c: a,b
q)
q)/- Sum the elements of c
q)sum(c)
15
q)/-multiply each element by 10
q)c*10
10 20 30 40 50
```

2. Define a new empty list, d.

```
q)d:()
```

3. Redefine d to be an empty list of type integer.

```
q)d:`int$() /-or d:`int$d
```

4. Add 5 random elements to d.

```
q)d, 5?til 10
8 1 9 5 4
q)/-in place
q)d,: 5?til 10
```

5. Create a list e with one element.

```
q)/- singleton list
q)e:enlist(10)
q)/-or
q)e:(),10
q)e
,10
q)type e
7h
```

6. Create a list, *list*1, with 20 random values between 3 until 30. Perform various arithmetic operations on *list*1.

```
q)/-create list1 with 20 random values from 3 til 30
q)list1: 20?3+til 30
q)max list1
30
q)min list1
3
q)avg list1
17.3
q)list1[10]
21
q)list1[19]
15
q)3 5 7 11 13 17 in list1
111101b
q)list1 * 3
87 21 48 36 15 30 9 60 51 36 63 60 33 81 48 60 87 78 90 45
q) /- note to make list1 permanently *3, use:
q) list1*:3
q)
q)/- add to each element of list1 its index
q)list1+: til count list1
q)
q)/- find all the even numbers, how many are there?
q)/- use mod 2 to find even elements
q)list1 mod 2
1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0
q)/- then count the zeros
q)count where not list1 mod 2
6
q)/- or use count ... where ... = 0
q)0 = list1 mod 2
01100000000001100011b
q)count where 0 = list1 mod 2
6
q)
q)/-take the first 10 and cast to dates
q)/-use # to grab the required elements
q)`date$10#list1
2000.03.28 2000.01.23 2000.02.20 2000.02.09 2000.01.20 ..
q)/- dates stored as integers (days) from 2000.01.01
```

7. Create the matrix m

```
q)/-create the matrix m
q)m:(1 2 3;4 5 6; 7 8 9)
q)
q)/-get the middle col assign to a
q)a:m[;1]
q)
q)/-replace the middle row with a
q)m[1]:a
q)
q) /- transpose m
q)mm:flip m
q)
q)/-join an extra row to mm consisting all of 10s
q)mm,:3#10
q)
```



```
q)/-join another row, 11 12 13
q)mm,: 11 12 13
```

8. Create the nested list
   - Find the type of each of the rows.
   - Collapse the nesting of this list.

```
q)/-create the nested list
q) list2: (1 2 3; `a`c`b; 10 11 12 14f; 100011b)
q)
q)/- find the type of each of the rows
q)type each list2
7 11 9 1h
q)/-collapse the nesting
q)raze list2
1
2
3
`a
`c
...
```

9. Define the strings $s1$:"Hello" and $s2$:"World", and manipulate as required.

```
q)s1:"Hello"
q)s2:"World"
q)
q)/-join them together
q)s:s1," ",s2
q)s
"Hello World"
q)
q)/-find the index of "W"
q)s?"W"
6
q)/-find the index of the last "l" in s
q)last ss[s;"l"]
9
q)/-remove hello from s and add of warcraft on the end in one operation
q)ssr[s;"Hello ";""], " of Warcraft"
"World of Warcraft"
```

10. The data regards the usage of some cool computer games from just a few years ago by our office team. Make lists from the data, and manipulate appropriately to get the required data.

```
q)/- computer games
q)games:`crash1`streets`echo`crash2`sonic`mm`pokemon`mario`bomber`zelda
q)platform:`ps1`sega`sega`ps1`sega`ps1`gameboy`gameboy`sega`gameboy
q)level:(7 9 6 8 4 1i; 2 1 3 5i; 8 5 7 5 4 4i;10 2 1; 4 2 1 10i;
8 4 9 5 8 10i;0 3 2 10 8 5 6 10i;6 0i;8 4 8 9 5 1 7i;0 10 9 5 2i)
q)
q)/- create number users list
q)show users:count each level
6 4 6 3 4 6 8 2 7 5
q)
```



```
q)/- avg level for each game
q)avg each level
5.833333 2.75 5.5 4.333333 4.25 7.333333 5.5 3 6 5.2
q)
q)/-boolean where avg more than 6
q)where 6 < avg each level
,5
q)
q)/- which games are more than 6 avg?
q)games where 6 < avg each level
,`mm
q)
q)/- sum for each platform
q)/- for later {sum users where platform in x} each distinct platform
q) sum users where platform in `ps1
15
q) sum users where platform in `sega
21
q) sum users where platform in `gameboy
15
```


Chapter 3

Dictionaries

1. Create a dictionary with the letters from a to z, with their corresponding numerical value

```
q)dict:.Q.a!1+til 26
q)
q)/-index in to find the numberical value of your name
q)sum dict "yourname"
112
q)/-rename the keys of the dictionary to be the uppercase letters
q)dict:.Q.A! value dict
```

2. Using another dictionary of your own design, change the uppercase words "HELLO", "WORLD" to lowercase.

```
q)dict2:.Q.A!.Q.a
q)dict2 "HELLO"
"hello"
q)dict2 "WORLD"
"world"
```

3. Create a dictionary morse, which contains the letters of the alphabet from n through to s as the keys and the morse code for those letters as the values. (Indicate the values as you like; maybe 1s and 0s?)

```
q)morse:("nopqrs")!(10b;111b;0110b;1101b;010b;000b)
q)/- add on numbers 0 to 4
q)morse,:("01234")!(11111b;01111b;00111b;00011b;00001b)
q)morse "sos"
000b
111b
000b
```

4. That was quite a topical question for Belfast. Here's another one, about the weather. Create two dictionaries, which will contain the following data with the places as keys, and one dictionary each for each year:

```
q)/-weather. it rains a lot in belfast, but more in tobermory
q)/- (it's a place on the west coast of Scotland)
```

```
q)places:`lon`edin`bfs`man`tob`port`car
q)R13:557 704 944 867 1681 674 1152
q)R12:600 854 1020 955 1789 544
q)D13:places!R13
q)D12:(-1_places)! R12
q)
q)/- extract the keys from the first
q)key D13
`lon`edin`bfs`man`tob`port`car
q)
q)/-values from the first
q)value D13
557 704 944 867 1681 674 1152
q)
q)/- sum rainfall
q)D12+D13
lon | 1157
edin| 1558
bfs | 1964
man | 1822
tob | 3470
port| 1218
car | 1152
q)/- avg rainfall for each city
q)avg (D12;D13)
lon | 578.5
edin| 779
bfs | 982
man | 911
tob | 1735
port| 609
car | 576
q)
q)/- annual rainfall more in 2013?
q)D13 > D12
lon | 0
edin| 0
bfs | 0
man | 0
tob | 0
port| 1
car | 1
q)
q)/- avg rainfall over the two yrs for the uk
q)sum (D12+D13) % 2
6170.5
```

5. Create a dictionary containing all the variables in the workspace as the key, and their values as the values of the dictionary.

```
q)/- create a dictionary of variables:values in workspace
q)/- try also value `. - all the vars and functions are already contained in a
    dict
q)vars:(system"v")!(value each (system"v"))
q)vars
D12     | `lon`edin`bfs`man`tob`port!600 854 1020 955 1789 544
D13     | `lon`edin`bfs`man`tob`port`car!557 704 944 867 1681 674 1152
R12     | 600 854 1020 955 1789 544
R13     | 557 704 944 867 1681 674 1152
a       | 2 5 8
...
```


Chapter 4

Table Operations

1. Create three lists, a, b, and c, each with 4 elements.
   - Make a new table, t with each column consisting of the lists a, b, and c.
   - Make a dictionary d from the table t.

```
q)/- create three lists a,b,c with 4 elements
q)a: 1 2 3 4
q)b:`a`b`c`d
q)c:10 11 12 13
q)
q)/- make a table t with a,b,c
q)t:([] a;b;c)
q)
q)/- make a dict from t
q)dict:flip t
```

2. Create an empty *trade* table, with 4 columns:

```
q)trade:([] sym:`$();side:`char$();size:`int$();price:`float$())
```

3. Create another empty table, *lasttrade*, which is a copy of *trade*. Key this by sym.

```
q)lasttrade:([sym:`$()] side:`char$();size:`int$();price:`float$())
```

4. Is there a difference in the metadata between *trade* and *lasttrade*? What about the type? What do each of the columns in meta represent?

```
q)(meta trade) ~ (meta lasttrade)
1b
q)(type trade) ~ (type lasttrade)
0b
```

5. Using three join commands add a row each time containing appropriate data to *trade*

```
q)trade ,: (`IBM;"B";100i;20f)
q)trade ,: (`MSFT;"S";80i;15f)
q)trade ,: (`AAPL;"S";150i;11.2f)
```

6. Using a single join operation add 3 more made-up rows into *trade*

```
q)trade ,: ([]sym:`AAPL`MSFT`IBM;side:string each(`BSS);
size:(90 77 150i);price:(10.8 19.5 11.2f))
```

7. Fill the table *lasttrade* with the data from *trade*.

```
q)/- fill the table lasttrade with trade
q)lastrade,:trade
```

8. The manager at a local shop wants to create a database of orders that he can view and get information from easily. Create and manipulate accordingly this table.

```
q)/- Stock table
q)/-create the table stock
q)stock:([] item:`soda`bacon`mush`eggs;
 brand:`fry`pork`veg`veg; price:1.50 1.99 0.88 1.55;
 order: 50 82 45 92)
q)
q)/- add the tomato row
q)stock,:(`tom;`veg;1.35;70)
q)/- view meta
q)meta stock
c    | t f a
-----| -----
item | s
brand| s
price| f
order| j
q)/-key the table
q)stock:`item`brand xkey stock
q)/- meta is the same - keys don't change that
q)meta stock
c    | t f a
-----| -----
item | s
brand| s
price| f
order| j
```

9. The manager decides to do his bulk orders together with a food stall trader at the market. They hope to save money that way. Create the *trader* table, and manipulate with the *stock* table accordingly.

```
q)/- trader - create table
q)
q)trader:([] item:`soda`bacon`mush`eggs`tom;
brand: `fry`pork`veg`veg`veg;price:1.5 1.99 0.88 1.55 1.35;
order: 200 180 110 210 100)
q)/-key the table by item and brand
q)trader:`item`brand xkey trader
```



```
q) /- create totalorders, dropping the original price col
q)totalorders: (2!(enlist `price)_0!trader) + 2!(enlist `price)_0!stock
q)
q)/-create newprices
q) newprices:0.75*stock`price
q)
q)/- join on the newprice which is 0.75 of the original price
q)totalorders:totalorders,'([] newp:newprices)
q)
q)/-how much better off are these clever merchants?
q)sum (stock`price) * (stock`order) *0.25
128.72
q)sum (trader`price) * (trader`order) *0.25
303.875
q)
```


Chapter 5

Functions

1. Write a monadic function, **f**, which calculates the square of a number (using an explicitly defined parameter).

```
q)f:{[a] a*a}
```

2. Execute **f**, with argument $a : 5$, and assign the result to new variable b.

```
q)a:5
q)b:f[a]
```

3. Write a dyadic function, **g**, which calculates the square of the first argument, divided by the square of the second argument (using implicit parameters).

```
q)g:{(x*x)%y*y}
```

4. Execute **g**, with arguments a and b. Store the result of this function in a variable called c.

```
q)c:g[a;b]
```

5. Create a dyadic function **f1** which indicates whether the product of two numbers is greater than their sum.

```
q)f1:{(x*y)> x+y}
```

6. Let $a = 5$, $b = a - 4$ and define $c = \frac{b}{a} + \frac{a}{b}$ using only one set of brackets.

```
q)a:5
q)b:a-4
q)c:(b%a) + a%b
```

7. Define a function **g1**, which calculates the polynomial $x^5 - 3x^2 + 5$ and determine the value of the polynomial when $x = 4, x = 8$ and $x = 6.7$.

```
q)g:{(x*x*x*x*x)+(-3*x*x)+5}
q)g:{(x xexp 5)+(-3*(x xexp 2))+5}
q)g[4]
q)g[8]
q)g[6.7]
```

8. Calculate the following without using brackets: $-5\left(\frac{6}{3-1}\right)+7$.

```
q)7+5*6%3-1
```

9. Create a function that calculates the area of a triangle. Find the area of a triangle of height 7cm and base 10cm.

```
q)f:{x*y%2}
q)f[7;10]
```

10. Create a function that calculates the sum of two squares of a number x. Find the sum of two squares of 18.

```
q)g:{2*x*x}
q)g:{2*(x xexp 2)}
q)g[18]
```

11. Create a function that takes three values a, b and c and calculates a^3+b^2+c. Find the value of the function at 13,3,6.

```
q)h:{[a;b;c](a*a*a)+(b*b)*c}
q)h:{[a;b;c](a xexp 3)+(b xexp 2)*c}
q)h[13;3;6]
```

12. A rough calculation of the BMI of a person takes their weight in kilograms and divides it by their height in metres squared. Create a formula which will calculate the BMI for any individual, without any explicit parameters.

```
q)BMI:{x%y*y}
```

13. Degrees Celcius is converted to degrees fahrenheit by multiplying by $\frac{9}{5}$ and adding 32, create a formula for this using only implicit parameters.

```
q)FAHREN:{32+9*x%5}
```

14. Using a parameter called perimeter, create a formula called area which will take the perimeter of a square and find it's area. This should only require one set of round brackets.

```
q)area:{[perimeter](perimeter%4)*perimeter%4}
```


15. **xexp** is a built-in dyadic function which calculates powers, for example 2 **xexp** 3 gives 8. Use this to re-create the formula in the previous question without explicit parameters.

```
q)area:{(x%4)*2}
```

16. A car depreciates at a rate of 15% for the first three and 8% for the next three years. Create a function k to find the value of a car after 6 years. Find the value of a car of starting value 15000GBP after 6 years.

```
q)f:{0.85*0.85*0.85*x}
q)g:{0.92*0.92*0.92*x}
q)g f[15000]
```

17. A function that a grocer can use each day to find out how much his fruit orders will cost.

```
q)total:{[apple;orange;banana;pear](apple*1.59)+(orange*1.99)
      +(banana*0.99)+pear*2.49}
```


Chapter 6

Scripts

6.1 Exercises from Notes: Answers

1. Writing a script 'inner.q' in order to:
   (a) Define a niladic function, **prnt**, to print "Loading inner" to the screen
   (b) Define a monadic function, **recip**, to return the reciprocal of a number.
   (c) Define a triadic function, **average3**, to return the average of 3 numbers.

```
inner.q will look like:

prnt:{show "Loading Inner"};
recip:{1%x};
average3:{(x+y+z)%3};
```

2. Writing a script 'outer.q' in order to:
   (a) Print out "Loading outer" to the screen.
   (b) Load script 'inner.q'
   (c) Run **prnt**.

```
outer.q will look like:

show "Loading Outer";
\l inner.q
prnt[]
```

3. Loading outer.q from the command line

```
q session will look like:

q)\l outer.q
"Loading Outer"
"Loading Inner"
```

   (a) Executing **average3** with arguments 3, 5, 10

```
q)average3[3;5;10]
6f
```

   (b) Executing **recip** with argument 3

```
q)recip[3]
0.3333333
```


6.2 Easy: Answers

1. Creating a script called 'shop' which generates the complete keyed table 'stock' from Exercises **??**

```
stock:([]item:`loaf`apples`milk`banana`cheese;
 brand:`breadwise`orchardfresh`dailydairy`orchardfresh`dailydairy;
 price_per_unit:1.00 1.5 .8 .99 5.;order:90 30 50 25 15);

save in a file called shop.q
```

2. Creating a script called 'HR' which generates the tables from Exercises **??**

```
country:([]id:1+til 4;city:`london`manchester`newyork`toronto;
 country:`uk`uk`usa`canada);

bonus:([]year:`int$();bonusvalue:`float$());

`bonus insert (1990 2000 2010;2000. 1500. 1000.);

bands:([id:"ABCD"];salary:("15-20";"20-30";"30-40";"40+"));
```

3. Using the ? function to generate an 'employee' script containing the 'employee' table. This includes employee id (primary key) and randomly generated marital statuses, city keys, salary band keys, genders and hire dates (before 2011). Edit the HR script to load this new employee's script.

```
In a file called employee.q put the following code:

employee:([id:til 150] marital:150?"MSDW";city:150?1+til 4;
 salary:150?"ABCD";gender:150?"MF";150?2011.01.01)

and in the HR.q file, include the following line:

\l employee.q
```

4. Loading 'shop.q' and viewing the stock table

```
\l shop.q
stock
```

5. Loading 'HR.q' and viewing the country and employee tables

```
\l HR.q
country
employee
```


6. Viewing the tables in the current q session

```
\a
```

7. Editing the previous scripts so that they show the names of all the tables available when they are loaded

```
add 0N!"loading filename.q" to beginning of each script
```


Chapter 7

Queries

7.1 Tables revision: Inserts and upserts

1. Creating the table

```
country:([]id:1 2 3 4;city:`london`manchester`newyork`toronto;
country:`uk`uk`usa`canada)
country
```

2. Creating an empty table called bonusvalue

```
bonus:([]year:`int$();bonusvalue:`float$())
bonus
```

3. Inserting data into the table

```
`bonus insert (1990 2000 2010;2000. 1500. 1000.)
 bonus
```

4. Creating a keyed table for salary bands called 'bands'

```
bands: ([code:`A`B`C`D];salary:("15-20";"20-30";"30-40";"40+"))
bands
```

5. Modifying the table bands

```
`bands upsert ([code:`D`E]salary:("40-60";"60+"))
 bands
```

7.2 Queries

1. Loading the fakedb.q script and creating a quotes/trades database

```
\l fakedb.q
makedb[10000;2000]
```

2. Calculating the aggr to be the total volume traded for each sym and side

```
aggr:select sum size by sym,side from trades
```

3. Modifying aggr sell volumes to be negative

```
update size:neg size from `aggr where side=`sell
```

4. Using aggr to calculate the final position for each symbol

```
select sum size by sym from aggr
**say positive is long, negative is short**
```

5. Storing all the IBM quotes as ibmqt

```
ibmqt:select from quotes where sym=`IBM
```

6. Adding a new boolean column, pricecross, to ibmqt to indicate whether the bid price is greater than or equal to the ask price

```
update pricecross:bid>=ask from `ibmqt
```

7. Finding the number of quotes where prices are crossed or not crossed from ibmqt

```
select count i by pricecross from ibmqt
```

8. Deleting all post-midday trades

```
delete from `trades where time>12:00:00.000
```

7.3 Easy

1. Loading the script 'shop'

```
\l shop.q
```

2. Finding from the stock table:

   (a) the item with the largest order

   ```
   select item,max order from stock
   ```

   (b) all items which are supplied by *OrchardFresh*

   ```
   select from stock where brand=`OrchardFresh
   ```

   (c) the brand with the maximum price

   ```
   select brand,max price_per_unit from stock
   ```


(d) the total number of orders

```
select sum orders from stock
```

(e) average order price according to brand

```
select avg price_per_unit by brand from stock
```

(f) total order size for each brand

```
select sum order by brand from stock
```

3. Creating a new column in stock called 'total_order_price', which is the product of 'price_per_unit' and 'order'.

```
update total_order_price:price_per_unit*order from `stock
```

4. Creating a new table called brands containing the 'brand' and 'total_brand_order_price' columns.

```
brands:select brand,total_order_price from stock
```

5. Loading the fakedb.q script and making a quotes/trades database

```
\l fakedb.q
makedb[10000;2000]
```

6. Finding the ratio of trades to quotes for syms and order the result from highest to lowest

```
ratiotab:{trdno:select trno:count[i] by sym from trades;
 qutno:select quno:count[i] by sym from quotes;
 `ratio xdesc update ratio:trno%quno from trdno lj qutno}
```

7. Finding the maximum time between each trade and the maximum time between each quote for the syms

```
maxt:{[x] trtab:(select difftimetr:max(time-prev time) by sym from trades);
 qutab:select difftimequ:max(time-prev time) by sym from quotes;
 select from (trtab lj qutab) where sym in x}

 maxt[`YHOO`ORCL]
```

8. Finding the sum bought and sold for each sym, and adding a field for the position based on the client side of the deal

```
midpr:{midtab:update mid:(bid+ask)%2 from aj[`sym`time;trades;quotes];
 select lmid:count i where price<mid,hmid:count i where price>mid,
 emid:count i where price=mid,mmid:max(abs(mid-price)) by sym from midtab}

 midpr[]
```


9. Finding the number of times the quote mid is up, down or stayed the same, and doing the same with trades in regard to price

```
mvqu:{select upno:count i where mid>prev mid,dono:count i where mid<prev mid,
 sano:count i where mid=prev mid by sym from update mid:(bid+ask)%2 from quotes
     }

mvtr:{select upno:count i where price>prev price,dono:count i
 where price<prev price,sano:count i where price=prev price
 by sym from trades}

mvqu[]
mvtr[]
```

10. Finding the number of trades executed inside the bid and ask, the number of buys above the mid and the number of sells below the mid in regards to sym

```
exetrd:{select match:count i where price = (bid|ask),
 buyabvmid:count i where (price>mid)&(side=`buy),
 sllblwmid:count i where (price<mid)&(side=`sell) by sym
 from update mid:(bid+ask)%2 from aj[`sym`time;trades;quotes]}

exetrd[]
```

11. Reversing the order of the characters of each sym in the trades

```
rvssym:{update `$(reverse each string[sym]) from trades}

rvssym[]
```

7.4 Moderate: Answers

1. Loading the school.q script and viewing the tables in q

```
In the current directory (outside of q), type q school.q and then \a
```

2. Viewing the marks for class A

```
select from marks where class="A"
```

3. Viewing the marks for the males of class B

```
select from marks where class="B",gender="M"
```

4. Finding the average french mark in class C

```
select avg mark from marks where subject=`french,class="C"
```

5. Displaying the average mark for each subject, ignoring classes

```
select avg mark by subject from marks
```


6. Finding the average, minimum, maximum and standard deviation of the maths marks for class A

```
select avg mark,min mark,max mark,dev mark from marks where
 subject=`maths,class="A"
```

7. Displaying the lowest, median and highest mark according to gender and subject

```
select min mark, med mark, max mark by gender,subject from marks
```

8. Finding how many people are in each class

```
select count distinct id by class from marks
```

9. Finding the median, range, sum of the marks and number of distinct marks from class B ict marks

```
select med mark,range:(max mark)-min mark, sum mark, count distinct mark
 from marks where subject=`ict, class="B"
```

10. Finding the highest average subject mark for class E, calling it topmark

```
select topmark:max mark from select avg mark by subject from marks
 where class="E"
```

11. Calculating the average mark for french in class D without using **avg**, then using **avg** to verify the result

```
select sum(mark)%count mark from marks where subject=`french,class="E"
```

12. Calculating the range of marks among the males in class D in ict and french

```
select range:(max mark)-min mark by subject from marks where class="D",
 subject in `ict`french, gender="M"
```

13. Deleting class E's results from the population

```
delete from `marks where class="E"
```

14. Modifying class A's french papers to add 5 marks to each of their scores

```
update mark:mark+5 from `marks where class="A"
```

15. Adding a new column called average to the table, which contains the average mark for that class and subject

```
update average:avg mark by subject,class from `marks
```

16. Loading the fakedb.q script and creating a quotes/trades database


```
\l fakedb.q
 makedb[10000;2000]
```

17. Finding the time where the price of each sym, from the trades table, changes by the largest absolute value for the list of syms provided

```
tmdiffp:{[x] dptab:update differp:abs(price-(prev price)) by sym from trades;
 select time where differp=max differp by sym from dptab where sym in x}

tmdiffp[`YHOO`ORCL]
```

18. Calculating the 10 and 15 trade moving averages for an individual sym, marking the instances (using boolean) where the values cross

```
across:{[x] mavgtab:select time,tenm:10 mavg price,fifm:15 mavg price
 from trades where sym in x;
 update across:(tenm>fifm)<>(prev tenm>fifm) from mavgtab}

 across[`YHOO]
```

19. Finding the number of times the trade price was below the mid price, above the mid price and equal to the mid price, and the absolute maximum difference between the mid price and the trade price for each sym

```
midpr:{midtab:update mid:(bid+ask)%2 from aj[`sym`time;trades;quotes];
 select lmid:count i where price<mid,hmid:count i where price>mid,
 emid:count i where price=mid,mmid:max(abs(mid-price)) by sym from midtab}

 midpr[]
```

7.5 Hard

1. Loading the fakedb.q script and creating quotes/trades database

```
\l fakedb.q
makedb[10000;2000]
```

2. Finding the average price of each sym in the five minutes after the hour, each hour, for the entire day in trades. Also adding to this table the average price of each sym in the last five minutes of each hour

```
avgp:{[x] ffive:select first5:avg price by sym,hour:time.hh from trades
 where (sym in x),(time mod 3600000)<=00:05:00.000;
 lfive:select last5:avg price by sym,hour:time.hh from trades
 where (sym in x),(time mod 3600000)>=00:55:00.000;
 ffive lj lfive}

 avgp[`YHOO`ORCL]
```

3. Finding a mark to market PNL and a position as well as the volume of all trades before a given time


```
mtmpnl:{[s;t]select from (update totvol:sums size,pos:sums size*?[side=`buy
    ;1;-1],
 MTM:(mid-price)*size*?[side=`buy;1;-1] by sym from (update mid:(bid+ask)%2
 from aj[`sym`time;trades;quotes])) where sym=s,time<t}

mtmpnl[`IBM;00:05:00.000]
```

4. Finding the price of an index at a particular time

```
tab1:flip `sym`size!(`IBM`YHOO`ORCL;1 2 3)

prcind:{[tab;t]select index:(sum tnp*size)%(sum t0p*size)%100
from ((tab lj (select t0p:first price by sym from trades))
lj (select tnp:last price by sym from trades where time<=t))}

prcind[tab1;00:05:00.000]
```


Chapter 8

Joins

Using the script 'fakedb.q' and the function 'makedb' to create tables.

1. Creating a 'quotes' table (of size 20)

```
makedb[20;0]
```

   (a) Storing the first 5 rows as 'q1'

```
q1:5#quotes
```

   (b) Storing the last 5 rows as 'q2'

```
q2:-5#quotes
```

   (c) Vertically joining 'q1' and 'q2' and storing as 'q3'

```
q3:q1,q2
```

2. Creating a 'quotes' tables (of size 1,000)

```
makedb[1000;0]
```

   (a) Calculating 'minbid' (the minimum bid value) and 'maxask' the maximum ask value for each symbol, and stores in a separate table

```
minmax:select minbid:min bid,maxask:max ask by sym from quotes
```

   (b) Left joins this table back to the original table

```
quotes lj minmax
```

3. Creating a 'quotes' and 'trades' tables (of size 1,000,000 and 200,000 respectively)

```
makedb[1000000;20000]
```

   (a) Joining, to each trade, the prevailing mid price (average of bid and ask) for the relevant time and sym.

```
aj[`sym`time;trades;select time,sym,mid:.5*bid+ask from quotes]
```


Chapter 9

Adverbs

1. If a is a list of lists, get the required parameters about a:

```
q)/ - a contains 3 lists:
q)count a
3
q)/ - to count the elements in each list use count each
q)count each a
2 1 3
q)/ - similarly use each to sum each list
q)sum each a
670 123 1303
q)/ - use raze to remove any nesting of lists
q)sum raze a
2096
```

2. Use the line of code:

```
prices:28+.01*floor 100*1000?5.0
```

   to generate a list of trade prices and assume that the order they appear in the list is the order in which they occurred. Find the total price of the largest two consecutive trades.

```
q)prices:28+.01*floor 100*1000?5.0
q)/-use each prior to sum each consecutive pair of trade prices
q)max +':[prices]
65.83 /-answer may be different as prices is randomly generated
```

3. The binomial coefficient $C(n,k)$ represents the number of ways of choosing k elements from a set of n elements. It can be given by the formula

$$C(n,k) = \frac{n!}{k!(n-k)!}.$$

   We can attempt to naively implement this in q by defining:

```
C:{[n;k] prd[1+til n]%prd[1+til k]*prd[1+til (n-k)]}
```

   Using this function find:

- The value of $C(15, k)$ where k ranges from 2 to 10.
- The value of $C(n, 3)$ where n ranges from 5 to 10.

```
q)C:{[n;k] prd[1+til n]%prd[1+til k]*prd[1+til (n-k)]}
q)/-Use each right to input multiple values for k
q)C/:[15;2 3 4 5 6 7 8 9 10]
105 455 1365 3003 5005 6435 6435 5005 3003f
q)/-Use each left for multiple values of n
q)C\:[5 6 7 8 9 10;3]
10 20 35 56 84 120f
```

4. Define the functions and varaibles and find which combination of atoms, lists and adverbs with f give the desired output.

```
q)f:{x+y*8}
q)a:1
q)b:4
q)c:3
q)list1: 1 2 3
q)list2: 10 9 8
q)list3: 3 7 1 5
q)/-each left
q)f\:[list3;b]
35 39 33 37
q)/-cross
q)f\:/:[list1;list2]
81 82 83
73 74 75
65 66 67
q)/-scan
q)f\[a;list2]
81 153 217
q)/-each right and scan
q)f/:[f\[c;list3];list3]
51 107 115 155
83 139 147 187
35 91  99  139
67 123 131 171
q)/-over
q)f/[list1;list3]
129 130 131
q)/-each each prior
q)f':[b;list3]
35 31 57 13
```


Chapter 10

Basic IPC

10.1 Exercises from Notes: Answers

1. Start a q server:

```
>q -p 1234
```

   Start a q client:

```
>q
```

   Define the following function on the q server:

```
q)f:{x*x}
```

   Run the function on the q server from the q client:

```
q)h(f;7)
```

2. Define this function on the q server:

```
q)f:{2*x}
```

   Run the function on the q server from the q client:

```
q)h(`f;7)
```

3. Use a negative handle to call the function asynchronously:

```
q)(neg h)(`f;7)
```

4. String form:

```
q)h"h:7"
```

   Parsed function form:

```
q)h(set;`h;7)
```

5. Set up the following fucntions on the server and client:

```
q)echo:{0N!x;}
q)add:{echo x+y}
```

Now define a function you want to use on the server, for example:

```
q)add3:{x+y+z}
q)proc:{ echo r:add3 . x; (neg .z.w) (y; r)}
```

Here the data for proc is passed as a list in the implicit parameter x, while the callback function name is passed as y . Note the use of . to evaluate a multi-valent function on a list of arguments. Now execute the following on the client

```
q)(neg h) (`proc; 1 2 3; `echo)
```

The result is that 6 is displayed on the server and then 6 is displayed on the client. For more examples visit http://code.kx.com/wiki/Cookbook/Callbacks

6. Send a sync chaser message to the remote:

```
q)h""
```

7. Use asynch flush:

```
q)neg[h][]
```

8. To multithread the input queue.

9. To keep a record of who is logging on:

```
q).z.po:{[x] show .z.w,.z.h}
```

To keep a record of who is logging off:

```
q).z.pc:{[x] show .z.w,.z.h}
```

10. To keep a record of incoming synchronous messages:

```
q).z.pg:{[x] show .z.w,.z.h,x}
```

To keep a record of incoming asynchronous messages:

```
q).z.ps:{[x] show .z.w,.z.h,x}
```

11. To serialize:

```
q)-8!{x*x}
0x0100000015000000064000a00050000007b782a787d
```


0x01 - little endian
000000
15000000 - message length
64 - type (100 - lamda)
00 - null terminated context (root)
0a - type (10 - char vector)
00 - attributes
05000000 - vector length
7b782a787d - x+y

12. To deserialize:

```
q)-9!0x0100000015000000064000a00050000007b782a787d
{x*x}
```

13. There is no ‘u$ when it is retrieved.

Chapter 11

Web Access

11.1 Exercises from Notes: Answers

1. Open port 9001.

```
q -p 9001
\p 9001           /if already in the q session
```

   (a) Creating a table with four fields and 5 entries named 'tab1'.

```
http://localhost:9001/?tab1:([]a:1 2 3 4 5;b:6 7 8 9 10;
c:11 12 13 14 15;d:16 17 18 19 20)
http://localhost:9001/?tab1
```

   (b) Defining a function that takes a table as an argument and returns the table in reverse order.

```
http://localhost:9001/?f:{[table] reverse table}
http://localhost:9001/?f[tab1]
```

   (c) Defining a function that takes a table and an integer as arguments and returns all the records in the table where the index number is less than the integer input.

```
http://localhost:9001/?g:{[tab;n]tab til n}
http://localhost:9001/?g[tab1;2]
```

2. (a) Extracting the table as a simple csv file.

```
http://localhost:9001/?save `:c:/q/tab1.csv
```

   (b) Extracting the table as an xml document.

```
http://localhost:9001/?save `:c:/q/tab1.xml
```

   (c) Loading the table into Microsoft Excel.

```
http://localhost:9001/.csv?tab1
```

11.2 Easy: Answers

1. (a) Start a q session and open port 4001

```
\p 4001
```

   (b) Creating the table called 'names', in the q session:

```
names:([]forename:(`jane;`clare;`john;`fred);
surname:(`brown;`smith;`white;`jones);
age:(32 24 61 63);sex:"FFMM")
```

   (c) Viewing the table in the browser window.

```
http://localhost:4001/
```

   (d) Using the browser window only:

      i. Select all forenames that are female.

```
http://localhost:4001/?select surname from names where sex="F"
```

      ii. Adding another entry to the table - a 20 year old woman called Anne Smith.

```
http://localhost:4001/?`names insert (`ann;`smith;20;"F")
```

      iii. Renaming Jane Brown as Sarah Brown

```
http://localhost:4001/?`names upsert(`brown;`sarah;32;"F")
```

      iv. Finding the average age of all entries in the table

```
http://localhost:4001/?select avg_age:avg age from names
```

      v. Creating a function which returns the gender of a given forename from 'names'.

```
http://localhost:4001/?select sex by forename from names
```

2. Assign the port 6789 to the current q session.

   (a) Executing the 'school.q' script.

```
http://localhost:6789/?\l school.q
```

   (b) View all tables

```
http://localhost:6789/?\a
```

   (c) Selecting all maths students from class B whose mark is greater than 70

```
http://localhost:6789/?select id from marks where class="B",
 subject=`maths,mark>70
```


(d) Finding the highest mark for each subject for male students

```
http://localhost:6789/?select highest_mark:max mark by subject
 from marks where gender="M"
```

(e) Finding the highest mark for each subject for female students

```
http://localhost:6789/?select highest_mark:max mark by subject
 from marks where gender="F"
```

(f) Finding the average mark in french and english for female students from class A.

```
http://localhost:6789/?select avg_mark:avg mark by subject
 from marks where class="A",gender="F",subject in`french`english
```


11.3 Moderate: Answers

1. Open the script fakedb.q on port 9001 using the command line prompt.

```
\p 9001
\l fakedb.q
```

2. View the variables present on a browser window.

```
http://localhost:9001/?\a
```

3. Making a trades and quotes table of length 1000000 and 200000 respectively.

```
http://localhost:9001/?makedb[1000000;200000]
```

4. Permanently adds a 'diff' column which is the positive difference in the ask and bid prices from the quotes table.

```
http://localhost:9001/?update diff:abs(bid-ask) from `quotes
```

5. Finding out how many quotes have a bid-ask spread greater than 0.05 for each symbol.

```
http://localhost:9001/?select number:sum (diff>0.05) by sym from quotes
```

6. Adding an hour onto each of the times.

```
http://localhost:9001/?update time:time +01:00:00.000 from quotes
```

7. Creating a function **f**

```
http://localhost:9001/?average_bid_col:select avg_bid:avg bid
by sym from quotes

http://localhost:9001/?quotes_extra:quotes lj average_bid_col

http://localhost:9001/?quotes_res:select result:bsize*avg_bid
from quotes_extra

http://localhost:9001/?select max result from quotes_res

http://localhost:9001/?select bids:sum bsize by sym from quotes
```


8. Changing the port to 9002 using the browser.

```
http://localhost:9001/?\p 9002
```

9. Correcting the corrupted CSCO data.

```
http://localhost:9002/?update bid:bid-0.05,ask:ask-0.05,
time:time-00:30:00.000 from quotes where sym=`CSCO
```

10. Removing asks and bids of less than 1000 units.

```
http://localhost:9002/?delete from `quotes where 1000>bsize
http://localhost:9002/?delete from `quotes where 1000>asize
```

11. Changing the DELL quotes to be AAPL.

```
http://localhost:9002/?update sym:`DELL from quotes where sym=`AAPL
```

12. Creating a new keyed table which is keyed by symbol and contains the median ask and bid prices.

```
http://localhost:9002/?select median_ask:med ask,median_bid:med bid
by sym from quotes
```


Chapter 12

Tables on Disk

Using the script 'fakedb.q' and the function 'makedb' to create tables.

1. Creating a directory (e.g. c:/dbex/), with 3 subdirectories 'flat', 'splay' and 'partition'

```
\mkdir dbex
\cd dbex
\mkdir flat
\mkdir splay
\mkdir partition
```

2. Creating quotes and trades tables (of size 10,000 and 1,000 respectively)

```
\l fakedb.q
makedb[10000;1000]
```

   (a) Storing 'quotes' as a flat table, without a name change, inside the 'flat' subdirectory

```
save `:C:/dbex/flat/quotes
```

   (b) Storing 'trades' as a flat table, called 'flattrades', inside the 'flat' subdirectory

```
`:C:/dbex/flat/flattrades set trades
```

3. Creating quotes and trades tables (of size 1,000,000 and 50,000 respectively)

```
\l fakedb.q
makedb[1000000;50000]
```

   (a) Deleting any symbol-type columns from 'quotes'. Stores as a splayed table (without a name change) in the 'splay' subdirectory

```
meta quotes
delete sym from `quotes
rsave `:C:/dbex/splay/quotes
```

(b) Storing 'trades' as an enumerated splayed table, named 'splaytrades', in the 'splay' subdirectory. Checks for a sym file.

```
`:C:/dbex/splay/tradesenum/ set .Q.en[`:C:/dbex/splay/;trades]
```

(c) Loading up the 'splay' subdirectory in a separate process and selects all IBM records from each table

```
\l C:\dbex\splay
```

4. Creating quotes and trades tables (of size 1,000,000 and 200,000 respectively)

```
\l fakedb.q
makedb[1000000;200000]
```

(a) Using .Q.dpft to store 'quotes' to the 'partition' subdirectory, with today's date, and extra partitioning by 'sym'

```
.Q.dpft[`:C:/dbex/partition/;.z.D;`sym;`quotes]
```

(b) Using .Q.dpft to store 'trades' to the 'partition' subdirectory, with today's date, and extra partitioning by 'sym'

```
.Q.dpft[`:C:/dbex/partition/;.z.D;`sym;`trades]
```

5. Starting a separate process to inspect the partitioned database

```
\l C:/dbex/partition/
trades
quotes
```

6. Starting the process listening on port 4321

```
\p 4321
```

7. Inspects the data through a browser

```
http://localhost:4321
```

8. Creating quotes and trades tables (of size 1,500,000 and 300,000 respectively)

```
\l fakedb.q
makedb[1500000;300000]
```

(a) Using .Q.hdpf to store both tables to the 'partition' subdirectory, uses yesterday's date, adds extra partitioning by 'sym', informs historical process of the change

```
.Q.hdpf[4321;`:C:/dbex/partition/;.z.D-1;`sym]
```

9. Checking on 4321 that the updates have been loaded

```
\l C:/dbex/partition/
```


Chapter 13

ETL - Extract, Transform, Load

1. Executing the following command will load a .csv file into memory as a table.

```
quotes: 0:[("SFF";enlist",");`:csvexercise1.csv]
quotes
```

2. Uploading the .csv table into memory and adds column names.

```
trades: 0:[("TSFIS";",");`:csvexercise2.csv]
trades
            **deliberately wrong**
trades:flip `time`sym`price`amount`type !0:[("TSFIS";",");`:csvexercise2.csv]
trades   **right**
```

3. Selecting AAPL rows and stored as apples in memory.

```
apples: select from trades where sym=`AAPL
apples
```

4. Storing the apples table as a .csv file

```
save `:c:/q/apples.csv
```

5. Loading 'apples.csv' back into memory, leaving out the 'time' column and storing the 'sym' and 'type' columns as strings.

```
0:[(" *FI*";enlist",");`:apples.csv]
```

Chapter 14

Inter Process Communication

14.1 Connections

1. Any of these are valid.

```
h:hopen 80001
h:hopen `::8001
h:hopen `:localhost:8001
/- open a handle with a 2000 ms timeout
h:hopen(`:localhost:8001;2000)
```

2.

```
q)h"1+1"
2
```

3. The user file should look like this :

```
joe:pass1
bob:pass2
```

   Assuming the user file is called users.txt, then to restart the process do

```
q -p 8001 -u users.txt
```

4. You need to append the username and password to the connection string e.g.

```
q)h:hopen `::8001:joe:badpassword
'access
q)h:hopen `::8001:joe:pass1
```

5.

```
allowedusers:`bob`frank
.z.pw:{[user;pass] user in allowedusers}
```

6.

```
clienthandles:([handle:`int$()] ip:`int$(); host:`symbol$();
        user:`symbol$(); opentime:`timestamp$())
.z.po:{`clienthandles upsert (x;.z.a;.z.h;.z.u;.z.p)}
```

7.

```
.z.pc:{delete from `clienthandles where handle=x}
```

14.2 Synchronous Calls

1. The three forms that can be used are the string form and the two forms using the function name.

```
q)h"add[5;8]"
13
q)h(`add;5;8)
13
q)h("add";5;8)
13
```

2. You can use either of the lambda forms.

```
q)multiply:{x*y}
q)h(multiply;10;20)
200
q)h("{x*y}";10;20)
200
```

3.
```
queries:([] time:`timestamp$(); handle:`int$(); ip:`int$();
 user:`symbol$(); query:(); accepted:`boolean$())

.z.pg:{
 /- log the incoming query
 `queries insert (.z.p; .z.w; .z.a; .z.u; -3!x;1b);
  /- evaluate it
  value x}
```

4.
```
queries:([] time:`timestamp$(); handle:`int$(); ip:`int$();
 user:`symbol$(); query:(); accepted:`boolean$())

.z.pg:{
 /- check if the incoming query is a stored procedure
 /- Stored proc calls should be mixed type lists
 /- with the first element being a symbol
 allowed:$[0h=type x; -11h=type first x; 0b];
 /- log the incoming query
 `queries insert (.z.p; .z.w; .z.a; .z.u; -3!x;allowed);
  /- if allowed, evaluate it, otherwise throw an error
  $[allowed; value x; '`$"stored procedures only"]}
```

14.3 Asynchronous Calls

1. No answer
2.
```
execute:{(neg .z.w)value x}
```

3. If execute is defined on the server, you can do

```
q)(neg h)(`execute; "1+1")
'type
```


If execute is defined on the client, you can do

```
q)(neg h)(execute; "1+1")
'type
```

4. Assuming execute is defined on the server:

```
q)(neg h)(`execute; "1+1"); r:h[];
q)r
2
```

5.

```
/- dictionary to store requests
requests:()!()

/- execute function simply inserts incoming request to dict
execute:{requests[.z.w]:x}

flushqueue:{
 if[count requests;
  /- execute the function, return the result back down the handle
  (neg first key requests)value first value requests;
  /- drop off the first request
  requests::1 _ requests]}
```

6. For simplicity, we will return an error string with a prefix of "error: "

```
flushqueue:{
 if[count requests;
  /- execute the function, return the result back down the handle
  (neg first key requests)@[value;first value requests;{"error: ",x}];
  /- drop off the first request
  requests::1 _ requests]}
```

7.

```
/- store both the request and the callback
execute:{[callback;query] requests[.z.w]:(callback;query)}

flushqueue:{
 if[count requests;
  request:first value requests;
  handle:first key requests;
  /- execute the function, pass the result back down the handle
  /- to the specified callback function
  (neg handle)(request[0];@[value;request[1];{"error: ",x}]);
  /- drop off the first request
  requests::1 _ requests]}
```

For a more fault tolerant version, you have to error trap the call back on the async handle as well - the client handle may have closed before it is executed.

```
flushqueue:{
 if[count requests;
  request:first value requests;
  handle:first key requests;
  /- execute the function, pass the result back down the handle
  /- to the specified callback function
  /- need to error trap the call back down the async handle as well
```



```
  @[neg handle;(request[0];@[value;request[1];{"error: ",x}]);()];
  /- drop off the first request
  requests::1 _ requests]}
```

To test it from the client, you can do

```
q)(neg h)(`execute;`processresult;"1+1")
q) /- do some other stuff in here
q) /- when you execute flushqueue[] on the server,
q) /- the result will be sent back
q)2013.12.19T07:16:32.948 the result has been received: 2
```

8.

```
.z.pc:{requests::(enlist x) _ requests}
```

9. We only need the async flush here, don't need to chase it with a sync message.

```
do[10;(neg h)"-1 string .z.p"; (neg h)[]; system"sleep 1"]
```


Chapter 15

HDB Queries

15.1 Selects

1. EASY Write a function called tradeticks which takes 3 parameters of start date, end date and symbols. The function should extract the trade data for the date range and symbol list.

```
straight select - nothing fancy here!
also only select the columns we need
most important part is to remember the ordering of the where clause
date, then sym, then anything else
remember to put startdate and end date in parenthesis

tradeticks1:{[startdate;enddate;symbols]
        select date,sym,time,size,price
        from trades
        where date within (startdate;enddate), sym in symbols}
```

2. EASY Create a new function, tradeticks2, which has two extra parameters - start time and end time. The function should extract only those trades which fall within the time range i.e. if start time is 12:00 and end time is 14:00, it will only return trades which occur in that window. HINT: you will have to extract the time portion from the timestamp field i.e. time.time or 'time$time

```
As previous, but add in the time selection

tradeticks2:{[startdate;enddate;symbols;starttime;endtime]
        select date,sym,time,size,price
        from trades
        where date within (startdate;enddate), sym in symbols,
              time.time within (starttime;endtime)}
```

3. MED Create a new function, tradeticks3, which is the same as tradeticks1 but uses a start timestamp and end timestamp as the parameters. It should query across the date boundaries. For example, if you run a query from 2012.01.01D09:00 to 2012.01.05D08:00, it should include all the ticks within that period. Make sure that it uses the date partitions correctly - the date portion should be run first and separately from the timestamp query.

```
Need to do some casting and add time into the where clause

tradeticks3:{[startdatetime;enddatetime;symbols]
        select date,sym,time,size,price
        from trades
        where date within `date$(startdatetime;enddatetime),
              sym in symbols,
              time within (startdatetime;enddatetime)}
```

4. MED Create tradeticks4 which is an optimized version of tradeticks3. If the start timestamp is equal to the first second of the day, and the end timestamp is equal to the last second of the day, then it doesn't execute the where filter on the time column.

```
As previous, but need to check the inputs
There are a few ways to get the start-of-day value from a timestamp e.g.
1D xbar timestamp
`timestamp$00:00+`date$timestamp
To get the end time, subtract 1 from the start  of the next date

tradeticks4:{[startdatetime;enddatetime;symbols]
 $[(startdatetime=`timestamp$00:00+`date$startdatetime) and
   (enddatetime=-1+`timestamp$1+`date$enddatetime);
     /- can call tradeticks1 here
     tradeticks1[`date\$startdatetime;`date$enddatetime;symbols];
     /- Can call tradeticks3 here
     tradeticks3[startdatetime;enddatetime;symbols]]}
```

5. HARD Create tradeticks5, which is as tradeticks4 except it additionally checks the type of the input. If a date type is passed in for both start date and end date, then it only executes the where on the date column. If the start and end values aren't of the same time (either both timestamps or both dates) then it should throw an error.

```
tradeticks5:{[start;end;symbols]
 /- Throw an error if the start and end types aren't both
 /- timestamp or date or if they don't match
 if[not (t:type start) in -12 -14h;
  '"start and end must be of type timestamp or date"];
 if[not t=type end; '"start and end must be of the same type"];
  $[t=-14h;
     tradeticks1[start;end;symbols];
     tradeticks4[start;end;symbols]]}
```

6. MED Create a function, quotechanges, which selects from the quote data for a given date list and symbol, only the data where either the bid price changes or the ask price changes.

```
remember to use () around the first part of the or statement
otherwise it executes as differ (bid1 or (differ ask1))

quotechanges:{[dates;symbols]
        select
        from quotes
        where date in dates,sym in symbols,(differ bid) or differ ask}
```


7. HARD Create a function which allows you to accumulate together bid quantities from the depth table. For example

```
accumdepth:{[d;s;l] ...}
```

will return all ticks for dates d and symbols s. Columns returned will be date, time, symbol, and accumdepth which is the depth sizes accumulated, specified by the parameter l. So if l is 2, accumdepth will contain bidsize1+bidsize2, if it is 3 it will be bidsize1+bidsize2+bidsize3. You will need to use a functional select for this.

```
Idea is to use a functional / dynamic query.
Best to use parse to see what it should look like

parse"select time,sym,totalsize:sum(bsize1;bsize2;bsize3) from depth
where date in d, sym in s"

returns

(?;`depth;,((in;`date;`d);(in;`sym;`s));0b;
 `time`sym`totalsize!(`time;`sym;(sum;(enlist;`bsize1;`bsize2;`bsize3))))

So all we need to do is build up the bsizeX list using the supplied
variable. This is just `$"bsize",/:string 1+til l
Note that the symbol variable s has to be enlisted here.

accumdepth:{[d;s;l]
 eval (?;`depth;enlist((in;`date;d);
      (in;`sym;enlist s));0b;
      `time`sym`totalsize!(`time;`sym;
        sum,enlist enlist,`$"bsize",/:string 1+til l))}
```

15.2 Aggregations

1. EASY The spread for a quote is defined as the difference between the bid and ask price. The ask price should always be higher than the bid price. Write a function which calculates the average spread between bid and ask per date and symbol from the quote table for a given date range and symbol list.

```
avgspread:{[startdate;enddate;symbols]
 select avgspread:avg ask-bid
 by date,sym
 from quotes
 where date within (startdate;enddate), sym in symbols}
```

2. EASY A common operation is to calculate High price, Low price, Open price, Close price values for each stock for each day. Create a function, dailystats1, which calculates the HLOC values for a given date range and list of instruments

```
HLOC buckets by date and symbol

dailystats1:{[startdate;enddate;symbols]
 select high:max price, low:min price, open:first price,
        close:last price
```



```
 by date,sym
 from trades
 where date within (startdate;enddate), sym in symbols}
```

3. EASY The VWAP is the Volume Weighted Average Price. kdb+ has a built in function, wavg, for calculating weighted averages. Create dailystats2, which is similar to dailystats1 but also calculates the VWAP value.

```
As previous but add the vwap

dailystats2:{[startdate;enddate;symbols]
 select high:max price, low:min price, open:first price,
        close:last price, vwap:size wavg price
 by date,sym
 from trades
 where date within (startdate;enddate), sym in symbols}
```

4. EASY The date can also be bucketted by time period using the xbar function. Create dailystats3 which is as dailystats2, but includes a bucketting value of type timespan.

```
As previous, but with bucketing

dailystats3:{[startdate;enddate;symbols;bucket]
 select high:max price, low:min price, open:first price,
        close:last price, vwap:size wavg price
 by date,sym,bucket xbar time
 from trades
 where date within (startdate;enddate), sym in symbols}
```

5. MED Create dailystats4, which is the same as dailystats3 but make sure the OPEN price returned in each bucket is actually the open price. The opening price should always be the same as the close price of the previous bucket for the the same symbol. Is it?

```
As previous, but change the open price to be the previous close

dailystats4:{[startdate;enddate;symbols;bucket]
 /- Update the opening price to be the previous close
 /- fill against the open price to handle where there
 /- is no previous close (i.e. the first bucket)
 update open:open^prev close
 by sym
 from dailystats3[startdate;enddate;symbols;bucket]}
```

6. HARD Create dailystats5, which is the same as dailystats4 but uses a starttimestamp and endtimestamp instead of startdate and enddate. You need to create a fully contiguous timeseries. If there is no data in a time bucket, then dailystats4 will not return any data. dailystats5 will include all time buckets. This is done by creating a "rack" table which contains all the combinations of symbols and required timebuckets, then filling against it. In this case, the value for each unpopulated HLOC value should be the same as the close price for the previous bucket (as the closing price carries forward).


```
dailystats5:{[starttimestamp;endtimestamp;symbols;bucket]
 /- pull out the required values in the same way as before
 /- except using an additional time clause
 stats:select high:max price, low:min price, open:first price,
              close:last price, vwap:size wavg price
        by date,sym,bucket xbar time
        from trades
        where date within `date$(starttimestamp;endtimestamp),
         sym in symbols,
         time within (starttimestamp;endtimestamp);

 /- build the list of times
 times:(bucket xbar starttimestamp)+
        bucket*til `long$1+(endtimestamp - starttimestamp)%bucket;
 /- build the rack based on symbol and timestamp
 rack:([]sym:symbols,()) cross ([]time:times);

 /- join against the rack
 res:rack#`sym`time xkey delete date from stats;

 /- fill forward all the close prices
 res:update fills close by sym from res;

 /- update the open price as per dailystats4
 /- fill sideways for the other values
 /- don't fill vwap as it should be null if there wasn't any trading
 update open:open^prev close, high:close^high,
        low:close^low by sym from res}
```

7. MED The TWAP (time-weighted average price) is similar to the VWAP, but instead of weighting the price by the size, the price is weighted by the length of time it existed as the last price. Create a function, vwapandtwap1 which calculates the VWAP and TWAP per day and per symbol for a supplied list of dates and symbols.

```
VWAP is just the price weighted by size, so quantity wavg price
TWAP is time weighted average price.
For TWAP, need to get the time difference between ticks and
multiply by the price.
(deltas time) will give the difference between consecutive times.
So you will need to use next to move it along one.
At each point it's then the difference between the current value
and the next.
With a TWAP, the last value doesn't count - as we don't know when
the next trade is, so don't know the weighting

twapandvwap1:{[startdate;enddate;symbols]
 select vwap:size wavg price, twap:(next deltas time) wavg price
 by date,sym
 from trades
 where date within (startdate;enddate), sym in symbols}
```

8. HARD Create twapandvwap2, which is as twapandvwap1 but calculates the difference on a

```
Use an fby to calculate the maximum disparity per date

twapandvwap2:{[startdate;enddate;symbols]
```



```
 select from
  (update diff:100*abs 1-twap%vwap
   from twapandvwap1[startdate;enddate;symbols])
  where diff=(max;diff)fby sym}
```

9. MED Create bigdays, which returns across a one month window any days where the volume of shares traded was greater than n stddevs from the average for that month.

```
Need to select out the total quantity traded on each day
then calculate the average and dev, and find any outliers.
Do it for all syms.
Pass a month as the parameter.

bigdays:{[mnth;stddevs]
 select from
  (select tradecount:sum `long$size
   by date
   from trades where date.month = mnth)
  where (abs tradecount-avg tradecount)>stddevs*dev tradecount}
```

10. HARD Create a function, iqr, which takes a list of data and calculates the interquartile range. (Don't have to be too precise about the calculation of the quartiles - an approximate number will do). Use the iqr function to the interquartile range of price for all symbols on any given day.

```
There are lots of ways to calculate interquartile range.
Here are a couple:

iqr:{neg (-) . avg each x(floor .25 .75*\:-1 0+count x:asc x)}
iqr2:{(-) . x rank[x]?`long$.75 .25*count x}

iqrs1:{[startdate;enddate;symbols]
 select range:iqr[price]
 by date,sym
 from trades
 where date within (startdate;enddate), sym in symbols}
```

11. EASY Create iqrs2, which will calculate for a date range and each symbol, which day had the largest iqr.

```
iqrs2:{[startdate;enddate;symbols]
        select
        from iqrs1[startdate;enddate;symbols]
        where range=(max;range) fby sym}
```

12. MED I want to xbar some data in 25 minute buckets. However, my trading day doesn't start until 9am and I want to make sure that 9am -> 09:25am is my first bucket. How can I do that? Create a function to do it. Note that in this function the bucket value should be an int or long rather than a timespan like it was in the previous questions.

```
We just have to move the start time of the bucket by subtracting
the start time, then adding it back on.
```



```
bucket:{[startdate;enddate;starttime;endtime;bucket;symbols]
 select sum size
 by date,sym,starttime+bucket xbar time.minute - starttime
 from trades
 where date within (startdate;enddate), sym in symbols,
       time.time within (starttime;endtime)}
```

13. MED I want to bucket some data in variable width buckets. My bucket start points are 08:00 08:15 08:32 08:50 09:27 12:00. How do I do that? (hint - use bin)

```
Index your time column into the list of bucket times using bin.
Then use that index to pull out the appropriate bucket, and
group on that.

b bin time.minute - returns an index into the bucket list
b b bin time.minute - maps the index back to the bucket value

variablebucket:{[startdate;enddate;symbols;buckets]
 select sum size
 by date,sym,buckets buckets bin time.minute
 from trades
 where date within (startdate;enddate), sym in symbols,
       time.time >= first buckets}
```

14. HARD Create a function to split the trading day into even volume buckets. So for example, if we wanted to split each trading day into 10 even buckets in terms of cumulative quantity traded, whow much volume would be traded in each bucket and what would the start and end times of the bucket be? HINT: look at xrank. Also, only write it to handle 1 symbol at a time.

```
The key here is to use xrank on the cumulative size value
everything else just falls out
volumebuckets1:{[startdate;enddate;symbol;numbuckets]
 select starttime:first time, endtime:last time, totalvol:sum size
 by date,volbucket:numbuckets xrank sums size
 from trades
 where date within (startdate;enddate), sym=symbol}
```

15. . MED Create volumebuckets2, which calculates the average start time and end time for each volume bucket across a set of dates. This will give the average volume profile. It should invoke volumebuckets1.

```
Should just invoke volumebuckets1, then average it
volumebuckets2:{[startdate;enddate;symbol;numbuckets]
 select starttime:`time$avg starttime.time,
        endtime:`time$avg endtime.time
 by volbucket
 from volumebuckets1[startdate;enddate;symbol;numbuckets]}
```

15.3 Calculated Data

These questions are aimed at taking raw data sets, and appending new calculated values to it.

1. MED Use tradeticks1 to select some data. Create a function to add in a column which computes the return of a price compared with the nth previous price.

```
Create a function to calculate the return using xprev

return:{[n;x] 100*x%n xprev x}

calcreturns:{[startdate;enddate;symbols;n]
 update ret:return[n;price]
 by sym
 from tradeticks1[startdate;enddate;symbols]}
```

2. MED Using the tradeticks1 function, create a function to extract data and add in running vwap and running twap columns. The running VWAP is the (running sum of the price*size)/running sum of size. The running TWAP is the (running sum of the price*active time)/running sum of active time.

```
Use update statements to add in the running columns
Better to calculate activetime once and use it twice

runningvwapandtwap:{[startdate;enddate;symbols]
 update rvwap:(sums size*price)%sums size,
        rtwap:(sums activetime*price)%sums activetime
 by date,sym
 from
        update activetime:next deltas time
    by date,sym
        from tradeticks1[startdate;enddate;symbols]}
```

3. MED Create a function to calculate the running sum of volume traded in each "price group" block. The function should use tradeticks1 to extract data from the trade table. A price group block is defined as a series of trades which are all executed at the same price.

```
The trick here is in the grouping
differ price returns a 1 when the price changes, and a 0 when not
sums differ price will calculate the running sum of the changes -
so those where no change will end up in the same group.

pricegroups:{[startdate;enddate;symbols]
 update pricegroupsize:sums size
 by sums differ price
 from tradeticks1[startdate;enddate;symbols]}
```

4. MED Extract a timeseries of data from the depth table (any date, symbol, time range you like). Add a column which displays the VWAP of all the levels on the bid, i.e. the VWAP you would get if you were to clear the depth for each update. Do the same for the ask.

```
Assuming 3 bid and ask columns, just manually work out the vwaps
wavg will vectorise across the columns

sidevwaps:{[startdate;enddate;symbols]
 update bidvwap:(bsize1;bsize2;bsize3) wavg (bid1;bid2;bid3),
        askvwap:(asize1;asize2;asize3) wavg (ask1;ask2;ask3)
```



```
 from
  select date,time,sym,bid1,bid2,bid3,bsize1,bsize2,bsize3,
         ask1,ask2,ask3,asize1,asize2,asize3
  from depth
  where date in (startdate;enddate),sym in symbols}
```

5. HARD Extract 30 minutes worth of depth ticks for any date and any symbol. Assuming I wish to sell 1000 shares, calculate for each tick how many levels down the book I will have to go to fill the order.

```
Couple of ways we could do this.
The more obvious way is to use bin,  If we use bin, it finds the
position in a sorted list where that element would go.

q)1.2 3.4 6.9 11.2 bin 3.7 12.0
1 3

So we can accumulate up depth sizes, then find how far down the book we have to
     go to fill our order.

findfilllevel:{[d;s;st;et;o]
 /- use sums to accumulate all the columns together, then
 /- use flip to transpose the bid size columns into a list of lists
 t:select date,time,sym,
          bsizes:flip sums(bsize1;bsize2;bsize3;bsize4;bsize5)
  from depth
  where date in d,sym in s,time within (st;et);

 /- Use the bin operator and \: adverb to apply to each element
 /- of the bsizes column in turn.
 /- We need to add 2 as bin starts counting from -1
 /- (if the order fills at the very start)
 update filllevel:2+bsizes bin\:o from t}

/- This is faster (no bin\: required, which could be slow)
findfilllevel2:{[d;s;st;et;o]
 select date,time,sym,
        filllevel:6-sum o<=sums (bsize1;bsize2;bsize3;bsize4;bsize5)
 from depth
 where date in d,sym in s,time within (st;et)}
```

6. HARD Similarly to above, assume I wish to trade 1000 shares. For each depth tick, calculate the average price I will receive for this trade size.

```
So, assuming we need to trade x shares, how far down the book
do we have to go and what price do we get?
The key to efficiency in this is to try to operate on whole
columns of data rather than resorting to adverbs.
The idea is to getting the running sum of the sizes then use &
to get the min of the order size and the cumulative volume
use deltas with this figure to get the volume traded at each price
then take a weighted average with the whole price

findfillprice:{[d;s;st;et;o]
 select date,time,sym,bid1,bid2,bid3,bsize1,bsize2,bsize3,
    fp:deltas[0;o&sums(bsize1;bsize2;bsize3)] wavg (bid1;bid2;bid3)
 from depth where date in d,sym in s,time within (st;et)}
```


7. EASY The following function

```
last5:{{-5 sublist x,y}\[();x]}
```

can be used to return the last 5 values at each point in a list. Try it on a list of integers and observe the result.

Use the tradeticks1 function defined and add a column containing the last 5 traded prices for each symbol.

```
If we wanted, we could parameterise the 5, so we have
lastx:{{neg[x] sublist y,z}[x;;]\[();y]}
e.g.
q)lastx[3] til 10

be careful of these this though, as it will increase memory usage

last5:{{-5 sublist x,y}\[();x]}
addlast5:{[d;s;starttime;endtime]
 update last5prices:last5 price
 by date,sym
 from tradeticks[d;s;starttime;endtime]}
```

8. MED Create a function to create a "rack" of data - a complete timeseries. This should take 4 parameters - a start timestamp, an end timestamp, a list of symbols, and a time bucket (type timespan). Use rack to select a complete bucketed timeseries from the trade table

```
Need to use start time, end time and bucket size to create the
correct time buckets.  Use cross to generate the rack

rack:{[starttimestamp; endtimestamp; symbols; bucket]
 /- build the list of times
 times:(bucket xbar starttimestamp)+
                bucket*til `long$1+(endtimestamp - starttimestamp)%bucket;
 /- build the rack based on symbol and timestamp
 rack:([]sym:symbols,()) cross ([]time:times);
 rack}

Can use a # to take the relevant items from the table
or could also use an lj (rack lj data)
only fill the price forward to the end of the given date

completeseries:{[starttimestamp;endtimestamp;symbols;bucket]
 update 0i^size
 from
  update fills price
  by time.date,sym
  from
   rack[starttimestamp;endtimestamp;symbols;bucket]#select
        sum size, max price
        by sym,bucket xbar time
        from trades
        where date within `date$(starttimestamp;endtimestamp),
              sym in symbols}
```

9. HARD Create a function to pivot the following table :


```
t:([] date:2010.01.01 _ til 6; sym:`A`B`C`A`B`C;
         price:10 11 15 20 4 11)
```

such that we have a matrix where date is the row identifier and symbol is the column identifier. (HINT : can build this up ourselves, or refer to code.kx.com)

```
This is pretty complex, but ok if we build it up
Create a list of the distinct symbols (the column headers)
q) u:exec distinct sym from t

Extract a dictionary mapping sym to price
q)exec sym!price from t
A| 10
B| 11
C| 15
A| 20
B| 4
C| 11

Use u# on the dictionary to ensure we have an entry for each header
q)exec u#sym!price from t
A| 10
B| 11
C| 15

Then do it by date - so one dictionary per date
q)exec u#sym!price by date from t
          | A  B  C
----------| --------
2010.01.01| 10 11
2010.01.02|       15
2010.01.03| 20 4  11

This is pretty much itâĂę except this result is a dict of
dicts, rather than a keyed table which is a dict of tables.
If we force a new name on the by clause, we get what we want
q)exec u#sym!price by date:date from t
date      | A  B  C
----------| --------
2010.01.01| 10 11
2010.01.02|       15
2010.01.03| 20 4  11

Then we can fill down (fills exec x) or zero fill (0^exec x) etc.
```

15.4 Joins

This section is about joining data between different tables

1. MED Create a function which takes as parameters a list of dates and a list of symbols. The function should extract the trade data and aj on the quote data, then calculate the number of trades which occurred on the bid, those which occurred on the prevailing ask, and those which occurred outside of the bid and the ask for each day.


```
With aj, we need to apply an attribute to the sym column the
recommendation when working with on disk data is just to use
attribute on disk
e.g. aj[`sym`extime;
        select from trade where date=d;
        select from quote where date=d]

However, if working with a small subset of syms it may be faster
to select the syms you want, then reapply the attribute.
It also may vary depending on whether the data is compressed
Additionally, it will probably be more efficient to process each
date at a time in an each loop.
But that may produce a slightly different result
i.e. in a 24 hour market we may need to carry prices across date
boundaries

Bottom line is, you should experiment!

whichside:{[datelist;symbols]
 select count i by time.date,sym,side
 from
  update side:?[price=bid;`bid;?[price=ask;`ask;`unknown]]
   from
   aj[`sym`time;
      select time,sym,price,size
      from trades
      where date in datelist, sym in symbols;
      update `g#sym from
       select time,sym,bid,ask
       from quotes
       where date in datelist, sym in symbols]}
```

2. MED Using the whichside function, create a function which calculates for any set of symbols symbol and over any list of days, which day had the largest disparity between the volume of shares executed on the bid and on the ask.

```
We need to do a sum of quantity per side of the book from the
table taken from whichside.
Then work out the difference between the sides.
We could do this without doing the intermediate "update" statement,
but that would mean duplicating the calculation of
last deltas quantity

tradedisparity:{[datelist;symbols]
 select
 from
  (update disparity:abs last deltas x
   by date,sym
   from
    select
    from whichside[datelist;symbols]
    where side in `bid`ask)
  where disparity=(max;disparity)fby sym}
```

3. HARD Create a function, adv, which takes 4 parameters - start date, end date, bucket size as a int (representing minutes) and symbol list. The function should calculate the average daily volume (cumulative) profile for the supplied parameters. Using this function to compare the trading which occurred yesterday with

the previous 20 trading days. Calculate, for each time bucket, the percentage of volume that had executed compared with the daily profile.

```
We need to sum up the totals sizes, then create the averages.
Can't just do average, as that will just compute the average
quantity in each bucket.
There's an assumption here that over a large date range, all the
buckets will be populated.
If that is not the case we will have to create a rack and fill against it.

adv:{[startdate;enddate;bucket;symbols]
 /- Cast to long so we don't go out-of-range when summing integers
 t:select total:sum `long$size
   by date, sym, bucket xbar time.minute
   from trades
   where date within (startdate;enddate), sym in symbols;

 /- Accumulate the totals for each day and symbol
 t:update sums total by date,sym from t;

 /- Sum up the totals, and divide by the distinct number of dates
 /- this is so we don't include weekends, holidays etc.
 /- Basically if there is data there, then it was a trading day
 /- if we don't need the number of days to be the same for all
 /- instruments, (i.e. instruments have different holidays),
 /- can just do
 /- select (sum total)%count distinct date by symbol,minute from t
 countdays:count exec distinct date from t;

 /- need to move each of the buckets forward, as xbar rounds the
 /- times down, but we want the minute to indicate the total volume
 /- traded by that point in time
 update minute:minute+bucket from
  select (sum total)%countdays
  by sym,minute
  from t}
```

4. MED Using the adv profile defined above, find the days within any 10 day period where the total volume traded by 12:00 was higher than the ADV for that period.

```
Can just use the ADV function twice.

comparetoadv:{[startdate;enddate;bucket;symbols;comparisondate]
 /- lj comparison data against the profile from startdate to enddate
 /- fill down the totals
 t:update fills total
   by sym
   from
    (`sym`minute`ADV xcol
     0!adv[startdate;enddate;bucket;symbols]) lj
       adv[comparisondate;comparisondate;bucket;symbols];
 update percentADV:100*total%ADV from t}
```

5. MED For one days worth of tick data, calculate the correlation of prices between two stocks which are supplied as a parameter. (this will require lining up the timeseries in some way)

```
Can line them up with either a uj (which will use all ticks for
both stocks) or and aj, which lines up the ticks of one stock with
```



```
the ticks of another stock.
Lets go with aj.

pricecorrelation:{[d;symbol1;symbol2]
 select correlation:cor[s1price;s2price]
 from aj[`time;
         select time,s1price:price
         from trades
         where date=d,sym=symbol1;
         select time,s2price:price
         from trades
         where date=d,sym=symbol2]}
```

6. HARD For each trade in the trade table on any given date, find the change in price for that symbol in the 5 minutes preceding the trade

```
To get the change in price for the previous 5 minutes, need to aj
the table onto itself.  Need to move the time forward by 5 minutes.

priceprev5:{[d;symbols]
 update pricechange:price - pminus5
 from aj[`sym`time;
         t;
         select time+0D00:05,sym,pminus5:price
          from t:select time,`g#sym,price
                 from trades
                 where date=d,sym in symbols]}
```

7. HARD For each trade in the trade table on any given date, find the vwap (for the symbol) for the 10 minutes following the trade

```
Do as above but use cumulative totals so we can work out the VWAP

vwapplus10:{[d;symbols]
 update vwapp10:(sumpsp10 - sumps)%sumszp10 - sumsz
 from aj[`sym`time;
         t;
         select time-00:10:00.0, sym,sumpsp10:sumps, sumszp10:sumsz
         from t:select time,sym,price,sumps:sums price*size,
                       sumsz:sums size
                from trades
                where date=d, sym in symbols]}
```

8. MED We are going to try to create a table containing all the bids / asks at +/- 1 2 5 10 minutes for every trade. Write a function to shift from quote the data at + x minutes. So, for example, if we were to query

```
shiftquote[quotedata;3]
```

it would return columns symbol, time,bidp3,askp3

where bidp3 and askp3 are the bid and ask in 3 minutes from the time column.

and if you were to query

```
shiftquote[quotedata;-3]
```


it would return columns symbol,time,bidm3,askm3

where bidm3 and askm3 are the bid and ask from 3 minutes previous.

```
shiftquote:{[q;x]
 (`time`sym,`$("bid";"ask"),\:(("p";"m")x<0),string abs x) xcol
  select time-x*0D00:01,sym,bid,ask from q}
```

9. HARD Use aj and / (over) to retrieve, for each trade, all the quotes at +/- 1 2 5 10 minutes. Use the shiftquote function

```
Create another function which calls getquote and ajs the result onto
a table t

ajquote:{[t;q;x] aj[`sym`time;t;shiftquote[q;x]]}

Create a wrapper to do the /
Only read the trades and quotes from disk once

plusminusquote:{[d;symbols;times]
 t:select date,time,sym,price,size
   from trades
   where date=d,sym in symbols;
 q:update `g#sym
   from select date,time,sym,bid,ask
        from quotes
        where date=d,sym in symbols;
 (ajquote[;q]/)[t;times]}
```


Chapter 16

Tickerplant

16.1 Tickerplant Setup

1. Download kdb+ tick. Unzip.
2. The schema should be as below. The 'g# attribute on sym is fairly standard to give speedups on RDB queries.

```
trades:([]time:`timespan$();sym:`g#`symbol$();
          price:`float$();size:`float$())
quotes:([]time:`timespan$();sym:`g#`symbol$();
          bid:`float$();ask:`float$();bsize:`float$();asize:`float$())
fxquotes:([]time:`timespan$();sym:`g#`symbol$();
          bid:`float$();ask:`float$())
```

3. To start the tickerplant:

```
q tick.q marketdata hdb -p 5010
```

   To start the RDB:

```
q tick/r.q :5010 -p 5011
```

4. No Answer

16.2 Feedhandler

1. No Answer
2. No Answer
3. No Answer
4. No Answer
5. The schema should be as below. The src column can be added in any position, as long as it matches the position in the FID map.

```
trades:([]time:`timespan$();sym:`g#`symbol$(); src:`symbol$();
          price:`float$();size:`float$())
quotes:([]time:`timespan$();sym:`g#`symbol$(); src:`symbol$();
          bid:`float$();ask:`float$();bsize:`float$();asize:`float$())
fxquotes:([]time:`timespan$();sym:`g#`symbol$();
          bid:`float$();ask:`float$())
```

In ssl1.q, you will need to amend the tv and qv lists which dictate the list of fields pushed to the database. Add FID 54 to this list.

```
/ trade and quote value fields - the fields we want for each message
tv:54 6,ti
qv:54,qi
```

6. The timestamp needs to be added before the symbol, in the k function.

```
k:{[s;x]
 /- if any of the trade FIDS are found, then it must be a trade
 if[any ti in key x;
        /- append the new data to the trade list
        t,:enlist (fi[18]@x 18),s,tf@'x tv]
 /- if any of the quote FIDS are found, then it must be a quote
 if[any qi in key x;
        q,:enlist (fi[18]@x 18),s,qf@'x qv];
 }
```

7. You will need to make a few modifications. You will have to add a variable to hold the fxquote data e.g.

```
t:q:fx:();
```

You will need to add FID maps and parsing functions

```
/ trade and quote value fields - the fields we want for each message
tv:54 6,ti
qv:54,qi
fv:22 25

/ dictionaries containing the parsing functions for each of the fids
/ tf = trade functions; qf = quote functions; ff = fx functions
tf:fi tv;
qf:fi qv;
ff:fi fv;
```

You will need to modify f to also publish the fx data, and clear out the variable when done.

```
f:{
 / k each x where"6"=x[;3];
 / x k'g each y;
 x k' y;
 if[count t;h(".u.upd";`trades;flip t)];
 if[count q;h(".u.upd";`quotes;flip q)];
 if[count fx;h(".u.upd";`fxquotes;flip fx)];
 t::q::fx::()}
```


And you will need to modify k, to parse out the fx values when an fx quote is received.

```
k:{[s;x]
 /- if any of the trade FIDS are found, then it must be a trade
 if[any ti in key x;
        /- append the new data to the trade list
        t,:enlist (fi[18]@x 18),s,tf@'x tv]
 /- if any of the quote FIDS are found, then it must be a quote
 if[any qi in key x;
        $[s in `AUDUSD`GBPUSD`EURUSD;
                fx,:enlist (fi[18]@x 18),s,ff@'x fv;
                q,:enlist (fi[18]@x 18),s,qf@'x qv]];
```

8. There are a few different ways this can be done. We will go for a generic approach, where we create a dictionary to cache all the previously observed values. Start by creating a structure to store the current values, and a function to update it.

```
/- all fid values, and all fid functions
/- af is a dict mapping fid!fidfunction
av:distinct 18,tv,qv,fv
af:av!fi av

/- dictionary to store the current values
/- this will be sym!(dictionary of fid!value)
currentvals:()!()

/- function to update the current cache
/- present is the list of FIDS present in the message which overlap
/- with the superset of required values
/- use this to index into both the incoming message,
/- and the dict of functions
/- apply each function to the relevant value
/- then update the current vals dictionary, and return the current vals
addtocurrent:{[s;x]
  currentvals[s] ,: present!af[present]@'x present:av inter key x;
  currentvals[s]}
```

Then we need to amend the k function, to update the current value cache and read values from it.

```
k:{[s;x]
 new:addtocurrent[s;x];
 /- if any of the trade FIDS are found, then it must be a trade
 if[any ti in key x;
        /- append the new data to the trade list
        t,:enlist new[18],s,new tv];
 /- if any of the quote FIDS are found, then it must be a quote
 if[any qi in key x;
        $[s in `AUDUSD`GBPUSD`EURUSD;
                fx,:enlist new[18],s,new fv;
                q,:enlist new[18],s,new qv]];
```

9. The schema should be changed to this :

```
trades:([]time:`timespan$();sym:`g#`symbol$();
          price:`float$();size:`int$())
quotes:([]time:`timespan$();sym:`g#`symbol$();
```



```
          bid:`float$();ask:`float$();bsize:`int$();asize:`int$())
fxquotes:([]time:`timespan$();sym:`g#`symbol$();
          bid:`float$();ask:`float$())
```

In the feedhandler, you will need to change the mapping functions which generate the types. This is simply done by changing the mapping functions for the appropriate FIDs.

```
i:6 22 25 178 30 31 379 1025 1067 393 270 18 54!
("F"$;"F"$;"F"$;"I"$;"I"$;"I"$;"V"$;"V"$;"V"$;"F"$;31="I"$;"n"$"P"$;"S"$)
```

When the RDB is restarted it will throw type an error such as

```
'type
!
-11
(27316;`:hdb/marketdata2013.11.21)
```

The problem that has occurred is that when replaying the tickerplant log file the RDB has encountered an error - the type of the supplied data does not match that expected in the schema. type or length errors are typical in the RDB when making schema modifications. A good solution is to modify the RDB upd function to catch the error. In r.q, change upd. The below function will suppress any errors. The fxquotes table should be populated after the replay, but the trades and quotes tables will be empty.

```
/upd:insert;
upd:{.[insert;(x;y);()]}
```

An alternative would be to trap the error but add some level of tracking to ensure the errors are registered e.g.

```
upderr:()!()
upd:{[tab;val] .[insert;(tab;val);{[tab;err] upderr[tab]+:1}[tab]]}
```

16.3 RDB and HDB Rollovers

1.
```
makehdb[`:tick/hdb/marketdata;10;10000;1000]
rm -r tick/hdb/marketdata/2013.11.21
```

2.
```
q hdb/marketdata -p 5012
```

To check the last date:

```
last date
```

To check the tables:

```
q)tables[]
`depth`trades`quotes
```


3.
```
q tick/r.q :5010 :5012 -p 5011
```

4. The last date in the database will now be the rolled down date. The list of tables should now be

```
q)tables[]
`fxquotes`trades`quotes
```

This is because kdb+ uses the most recent partition to dictate the list of tables available to it. fxquotes is in the most recent partition, depth is not.

5. The error will be on fxquotes, something like this.

```
q)select count i by date from fxquotes
k){$[#pm;pm[x]z;z in vt x;vp x;+(!+. x)!`/:dd[y;z],x]}
'./2013.11.08/fxquotes: No such file or directory
.:
`:./2013.11.08/fxquotes
```

The reason is because the fxquotes table only exists in the most recent directory, not the historic ones - i.e. you have just added it to the database. The solution is to use .Q.chk[hdbdirectory] which will back populate each of the tables, and forward populate any missing tables (i.e. depth). You can run .Q.chk in the HDB like this:

```
q).Q.chk`:.
:./2013.11.22
:./2013.11.21
:./2013.11.20
:./2013.11.19
:./2013.11.18
:./2013.11.15
:./2013.11.14
:./2013.11.13
:./2013.11.12
:./2013.11.11
:./2013.11.08
```

6. The available tables should now be

```
q)tables[]
`depth`fxquotes`trades`quotes
```

7. The schema should now look like this:

```
trades:([]time:`timespan$();sym:`g#`symbol$();
          price:`float$();size:`int$())
quotes:([]time:`timespan$();sym:`g#`symbol$();
          bid:`float$();ask:`float$();bsize:`int$();asize:`int$())
fxquotes:([]time:`timespan$();sym:`g#`symbol$();
            bid:`float$();ask:`float$())
depth:([]time:`timestamp$();sym:`g#`symbol$();
         bid1:`float$();bsize1:`int$();
         bid2:`float$();bsize2:`int$();
```



```
        bid3:`float$();bsize3:`int$();
        ask1:`float$();asize1:`int$();
        ask2:`float$();asize2:`int$();
        ask3:`float$();asize3:`int$())
```

8. There are a couple of ways to do this. Either modify .u.end directly, or create a wrapper which makes the modifications then invokes the old .u.end. Should probably start by making an adjusttime function to change the type of a column in the table

```
adjusttime:{[date;tablename]

 /- Check the time column is available
 /- and is one of the time types
 if[`time in cols tablename;
   if[(type (value tablename)`time) in 16 17 18 19h;

     /- update the column in place
     update time:`timestamp$date+time from tablename]]}
```

Then need to modify .u.end. We can either adjust .u.end directly in r.q e.g.

```
.u.end:{adjusttime[x] each tables; REMAINING DEFINITION}
```

or create a new definition which invokes the old definition. This is the more scalable approach. It means we either just have to add a line to the end of r.q or, better still, we can create an rdb.q script which is a wrapper around r.q, and modifies the definition of .u.end after r.q is loaded. This is better for upgrades - if r.q changes, we just have to update r.q rather than merge our updates with the new version of r.q.

```
.u.end:{[func;date]
 adjusttime[date] each tables[];
 func@date}[.u.end]
```

9. No Answer

10. The first part is to create a function to write down the daily table to atop level splay:

```
savedaily:{[currentdate]
 /- Save the table
 `:daily/ upsert .Q.en[`:.;0!select high:max price, low:min price,
                            open:first price, close:last price
                           by date:currentdate,sym from trades];
 /- Set the attribute
 @[`:daily;`date;`p#];
 }
```

Then add that to the end-of-day function:

```
.u.end:{[func;date]
 adjusttime[date] each tables[];
 savedaily[date];
 func@date}[.u.end]
```


16.4 Real Time Analytics

1. A possible script is:

```
/- Get the host:port from the command line
tickerplantport:`$.z.x 0

/- open a handle to the tickerplant
h:@[hopen;(tickerplantport;2000);0]

/- exit if no valid connection
if[not h;
 -2"cannot create connection to tickerplant on host:port ",
    string tickerplantport;
 exit 1]

/- subscribe
h(`.u.sub;`trades;`);

/- define a dictionary to maintain the counts
tradecounts:()!()

/- define the upd function to count the trades by sym
upd:{[t;x]
 if[t=`trades;
  /- Keep track of the number of trades
  tradecounts+::exec count i by sym from x]}
```

   Which can be run with

```
q subscriber.q ::5010 -p 5013
```

2. No Answer

3.

```
/- table to hold temporary vwap data
vwaps:([sym:`symbol$()] sumsize:`long$(); sumpricesize:`float$())

/- calculate the vwap
calcvwap:{[syms]
 select sym,vwap:sumpricesize%sumsize from vwaps where sym in syms}

/- define the upd function to count the trades by sym
/- and track the vwap
upd:{[t;x]
 if[t=`trades;
  /- Keep track of the number of trades
  tradecounts+::exec count i by sym from x;
  /- Track the VWAP
  vwaps+::select sumsize:sum`long$size,sumpricesize:sum price*size
          by sym from x]}
```

4. The necessary additions are to connect to the RDB, and build the initial variables from there. So for the connection :

```
/- And the RDB port
rdbport:`$.z.x 1

/- Open a handle to the RDB.
```



```
hr:@[hopen;(rdbport;2000);0]

if[not hr;
 -2"cannot create connection to RDB on host:port ",string rdbport;
 exit 2]
```

Then use the RDB to prime the internal variables:

```
/- populate tradecounts from rdb
tradecounts,:hr"exec count i by sym from trades"

/- populate vwaps from rdb
vwaps,:hr"select sumsize:sum`long$size,sumpricesize:sum size*price by sym from
    trades"
```

Run the process like this:

```
q subscriber.q ::5010 ::5011 -p 5013
```

5. The first step is to load u.q, declare the table, and initialise the pubsub.

```
/- load u.q
\l tick/u.q

/- define a vwap table to be published
currentvwap:([]time:`timestamp$(); sym:`symbol$(); vwap:`float$())

/- initialise pub/sub
.u.init[]
```

The next step is to modify the upd function to additionally publish the changed vwaps on each update:

```
/- define the upd function to count the trades by sym
/- and to accumulate and publish vwaps
upd:{[t;x]
 if[t=`trades;
  /- Keep track of the number of trades
  tradecounts+::exec count i by sym from x;
  /- Track the VWAP
  vwaps+::select sumsize:sum`long$size,sumpricesize:sum price*size
          by sym from x;
  .u.pub[`currentvwap;
         select time:.z.P,sym,vwap from calcvwap[distinct x`sym]]]}
```

6. We can just do this directly in the q console. No need for a script.

```
q)h:hopen 5013
q)upd:{show x; show y}
q)h(`.u.sub;`currentvwap;`)
`currentvwap
+`time`sym`vwap!(`timestamp$();`symbol$();`float$())
```

The updates should then start flowing in and printing to the screen.


```
q)`currentvwap
time                          sym  vwap
------------------------------------------
2013.11.22D15:40:34.124842000 MSFT 36.23361
`currentvwap
time                          sym  vwap
------------------------------------------
2013.11.22D15:40:34.124969000 MSFT 36.23349
`currentvwap
time                          sym  vwap
------------------------------------------
2013.11.22D15:40:35.123375000 AAPL 25.88361
```

16.5 Daemonizing

1. Windows:

```
start "Tickerplant" q tick.q marketdata hdb -p 5010
start "RDB" q tick/r.q :5010 :5012 -p 5011
start "HDB" q hdb/marketdata -p 5012
start "Analytics" q subscriber.q ::5010 ::5011 -p 5013
```

Linux:

```
q tick.q marketdata hdb -p 5010 </dev/null >tickerplant.log 2>&1 &
q tick/r.q :5010 :5012 -p 5011 </dev/null >rdb.log 2>&1 &
q hdb/marketdata -p 5012 </dev/null >hdb.log 2>&1 &
q subscriber.q ::5010 ::5011 -p 5013 </dev/null >hdb.log 2>&1 &
```

2. Something like below should suffice :

```
connection:hsym `$.z.x 0

/- open a connection
/- timeout the connection attempt after 2 seconds
h:@[hopen;(connection;2000);0]

$[not h;
 -2(string .z.z)," Failed to open connection to ",string connection;
 /- send an exit command down the handle, then flush the handle
 [neg[h]("exit 0"); @[h;(::);()]]];

exit 0
```

3. Windows:

```
start "KillTickerplant" q kill.q ::5010 -p 15010
start "KillRDB" q kill.q ::5011 -p 15011
start "KillHDB" q kill.q ::5012 -p 15012
start "KillAnalytics" q kill.q ::5013 -p 15013
```

Linux:

```
q kill.q ::5010 -p 15010 </dev/null >killtickerplant.log 2>&1 &
q kill.q ::5011 -p 15011 </dev/null >killrdb.log 2>&1 &
q kill.q ::5012 -p 15012 </dev/null >killhdb.log 2>&1 &
q kill.q ::5013 -p 15013 </dev/null >killhdb.log 2>&1 &
```


4. The kill script for a list of host:ports could look like :

```
connections:hsym each `$.z.x

openandkill:{
 /- open a handle
 h:@[hopen;(x;2000);0];
 $[not h;
  -2(string .z.z)," Failed to open connection to ",string x;
  /- send an exit command down the handle, then flush the handle
  [neg[h]("exit 0"); @[h;(::);()]]]}

openandkill each connections;

exit 0
```

and be invoked as

```
q killmany.q ::5010 ::5011 ::5012 ::5013
```


Chapter 17

File Loading

17.1 Basic Loading

1. No Answer
2. The load line is

```
/- load a single file into a big in-memory table
load1file:{[csvfile] ("DTSSFIFFIICS";enlist",")0: hsym csvfile}
```

   which is invoked like this :

```
load1file[`:csv/tradesandquotes20131125.csv]
```

3. This can be done be dropping the mapping to it

```
load1file:{[csvfile] ("DTSSFIFFII S";enlist",")0: hsym csvfile}
```

4.

```
load1file:{[csvfile]
 /- Load the data and rename the columns
 `time`sym`src`price`size`bid`ask`bsize`asize`curr xcol
  ("TSSFIFFII S";enlist",")0: hsym csvfile}
```

5.

```
load1file:{[csvfile]
 /- Load the data and rename the columns
 tab:`time`sym`src`price`size`bid`ask`bsize`asize`curr xcol
     ("TSSFIFFII S";enlist",")0: hsym csvfile}

 /- Modify the src field
 tab:update src:`$5 _' string src from tab;
 tab}
```

6.

```
load1file:{[csvfile]
 /- Load the data and rename the columns
 tab:`time`sym`src`price`size`bid`ask`bsize`asize`curr xcol
     ("TSSFIFFII S";enlist",")0: hsym csvfile;
```

```
 /- Modify the src field
 tab:update src:`$5 _' string src from tab;

 /- extract the date from the filename
 /- and add to the time field
 date:"D"$-8#-4 _ string csvfile;
 tab:update time:`timestamp$date+time from tab;

 tab}
```

7. We can create the tables as globals then use .Q.dpft to store. Remember to clear out the globals when done.

```
load1file:{[hdbdir;csvfile]
 /- Load the data and rename the columns
 tab:`time`sym`src`price`size`bid`ask`bsize`asize`curr
      xcol ("TSSFIFFII S";enlist",")0: hsym csvfile;

 /- Modify the src field
 tab:update src:`$5 _' string src from tab;

 /- extract the date from the filename
 /- and add to the time field
 date:"D"$-8#-4 _ string csvfile;
 tab:update time:`timestamp$date+time from tab;

 /- Store into global variables
 trades::select time,sym,price,size from tab where not null price;
 quotes::select time,sym,bid,ask,bsize,asize from tab where null price;

 /- Save down to the HDB
 .Q.dpft[hdbdir;date;`sym] each `trades`quotes;

 /- Clear out the globals
 delete from `trades;
 delete from `quotes;
 }
```

which can be invoked like this:

```
load1file[`:hdb;`:csv/tradesandquotes20131125.csv]
```

8. Binary tables can be saved keyed. To save in binary format you should use upsert with the table name e.g.

```
`:hdb/tablename upsert tabledata
```

Splayed tables cannot be keyed. To save in splayed format, you need to add a trailing forward slash to the table name, and also enumerate e.g.

```
`:hdb/tablename/ upsert .Q.en[`:hdb;tabledata]
```

The full code is

```
/- load a single file into a big in-memory table
load1file:{[hdbdir;csvfile]
 /- Load the data and rename the columns
```



```
tab:`time`sym`src`price`size`bid`ask`bsize`asize`curr
     xcol ("TSSFIFFII S";enlist",")0: hsym csvfile;

/- Modify the src field
tab:update src:`$5 _' string src from tab;

/- extract the date from the filename
/- and add to the time field
date:"D"$-8#-4 _ string csvfile;
tab:update time:`timestamp$date+time from tab;

/- Store into global variables
trades::select time,sym,price,size from tab where not null price;
quotes::select time,sym,bid,ask,bsize,asize from tab where null price;

/- Save down to the HDB
.Q.dpft[hdbdir;date;`sym] each `trades`quotes;

/- save down the static table
(`$(string hdbdir),"/static") upsert
      select last curr by sym from tab where not null price;
/- and the daily table
daily:select high:max price, low:min price, open:first price,
      close:last price by date:date,sym from trades;
(`$(string hdbdir),"/daily/") upsert .Q.en[hdbdir;0!daily];

/- Clear out the globals
delete from `trades;
delete from `quotes;
}
```

9. On our test machine the operation takes approximately 2 seconds and 200MB of RAM

```
q)\ts load1file[`:hdb1;`:csv/tradesandquotes20131125.csv]
1896 213388128
```

17.2 Loading Big Data Sets

1. No Answer

2. An example script would be:

```
load1filechunk:{[hdbdir;csvfile;csvdata]

 /- Load the data in, give the columns names
 tab:flip `time`sym`src`price`size`bid`ask`bsize`asize`curr
      !("TSSFIFFII S";",")0: csvdata;

 /- Modify the src field
 tab:update src:`$5 _' string src from tab;

 /- extract the date from the filename
 /- and add to the time field
 date:"D"$-8#-4 _ string csvfile;
 tab:update time:`timestamp$date+time from tab;

 /- Store into local variables
```



```
 trades:select time,sym,price,size from tab where not null price;
 quotes:select time,sym,bid,ask,bsize,asize from tab where null price;

 hdbbase:(string hdbdir),"/",string date;
 -1(string .z.Z)," writing ",(string count trades)," trades";
 (`$hdbbase,"/trades/") upsert .Q.en[hdbdir;trades];
 -1(string .z.Z)," writing ",(string count quotes)," quotes";
 (`$hdbbase,"/quotes/") upsert .Q.en[hdbdir;quotes];
 -1(string .z.Z)," complete";
 }
```

Which can be invoked as

```
q)load1bigfile[`:hdb;`:csv/tradesandquotes20131125.csv;read0`:csv/
    tradesandquotes20131125.csv]
2013.11.26T14:12:06.208 writing 100000 trades
2013.11.26T14:12:06.215 writing 1000000 quotes
2013.11.26T14:12:06.308 complete
```

3. An example is:

```
q)\ts load1filechunk[`:hdb;`:csv/tradesandquotes20131125.csv;read0`:csv/
    tradesandquotes20131125.csv]
2013.11.26T14:14:05.491 writing 100000 trades
2013.11.26T14:14:05.497 writing 1000000 quotes
3033 385524592
```

4.

```
.Q.fs[load1filechunk[`:hdb;`:csv/tradesandquotes20131125.csv];
      `:csv/tradesandquotes20131125.csv]
```

5. You should see that the memory usage is decreased, but the time is increased.

```
q)\ts .Q.fs[load1filechunk[`:hdb;`:csv/tradesandquotes20131125.csv];
           `:csv/tradesandquotes20131125.csv]
... debug output ....
4056 42872560
```

6. There is no correct answer here - it depends on the characteristics of your system. Generally though the default size of .Q.fs (128kB) is too small and should be increased.

7.

```
/- load a big file, using the chunk loader
load1bigfile:{[hdbdir;csvfile]
 /- invoke .Q.fsn with chunksize of 10MB
 .Q.fsn[load1filechunk[hdbdir;csvfile];csvfile;10000000]}
```

8.

```
load1bigfile:{[hdbdir;csvfile]
 /- invoke .Q.fsn with chunksize of 10MB
 .Q.fsn[load1filechunk[hdbdir;csvfile];csvfile;10000000];

 /- Get the HDB partition from the csvfile name
 hdbpart:(string hdbdir),"/",(string "D"$-8#-4 _ string csvfile),"/";

 /- Sort and part the data on disk
```



```
 {[d;t]
   -1(string .z.Z)," sorting and setting attribute for ",string t;
   `sym xasc tab:`$d,string t;
   /- Set the attribute
   @[tab;`sym;`p#];
 }[hdbpart] each `trades`quotes;
 }
```

9.

```
loadcsvs:{[hdbdir;csvdir]

 /- build the list of csv files
 csvfiles:`$((string csvdir),"/"),/:string files where
              (files:key csvdir:hsym csvdir) like "*.csv";

 /- iterate through and load each one
 {[h;f]
  -1(string .z.Z)," loading ",(string f)," to ",string h;
  load1bigfile[h;f]}[hdbdir] each csvfiles
 }
```

10. As the data is being appended to rather than overwritten, re-running the loading procedure causes the data to be duplicated.

Chapter 18

Creating a Trade Report

1. Assuming the command line paramters are passed in as `-dates 2014.04.21 2014.04.30`, assign them to the variable dates with:

```
dates:"D"$.z.x[1 2];
```

2. Protected evaluation of hopen:

```
h:@[hopen;2222;{-2"Failed to connect to HDB: ",x;exit 1}];
```

3. Synchronous query of HDB, passing in `dates` as argument:

```
trades:0!h({[DATES] select ntrades:count i, sum size,turnover:sum size*
    price by date,sym from trades where date within DATES};dates);
```

4. Load symstatic.csv:

```
static:("S*S";enlist ",") 0: `:tradereport/symstatic.csv;
```

5. Join static data:

```
trades:trades lj 1!static;
```

6. Load fxrates.csv:

```
fxrates:("DSF";enlist ",") 0: `:tradereport/fxrates.csv;
```

7. Join fx rates and update USD rate:

```
trades:aj[`curr`date;trades;update `g#curr from fxrates];
update rateUSD:1. from `trades where curr=`USD;
```

8. Create `tradereport` table:

```
tradereport:select date,sym,description,curr,ntrades,size,turnoverUSD:`
    long$turnover*rateUSD from trades;
```

9. Save report to csv:

```
save `:tradereport/tradereport.csv;
```

or

```
(`$":tradereport/tradereport_",(("_" sv string dates) except "."),".csv
    ") 0: "," 0: tradereport;
```

10. Add logging messages e.g.

```
-1 (string .z.p),": Connecting to HDB...";
```

18.1 Final Script

```
/define dates from command line variables
dates:"D"$.z.x[1 2];

/open port to the HDB process
-1 (string .z.p),": Connecting to HDB...";
h:@[hopen;2222;{-2"Failed to connect to HDB: ",x;exit 1}];

/query HDB
-1 (string .z.p),": Querying HDB...";
trades:0!h({[DATES] select ntrades:count i, sum size,turnover:sum size*price by date,
    sym from trades where date within DATES};dates);

/load symstatic.csv
-1 (string .z.p),": Loading symstatic.csv...";
static:("S*S";enlist ",") 0: `:tradereport/symstatic.csv;

/join static data to trades
-1 (string .z.p),": Joining static data...";
trades:trades lj 1!static;

/load fxrates.csv
-1 (string .z.p),": Loading fxrates.csv...";
fxrates:("DSF";enlist ",") 0: `:tradereport/fxrates.csv;

/join fxrates;
-1 (string .z.p),": Joining fxrates...";
trades:aj[`curr`date;trades;update `g#curr from fxrates];
update rateUSD:1. from `trades where curr=`USD;

/calculate turnover in USD
tradereport:select date,sym,description,curr,ntrades,size,turnoverUSD:`long$turnover*
    rateUSD from trades;

/save report to csv file
-1 (string .z.p),": Writing to csv...";
save `:tradereport/tradereport.csv;
(`$":tradereport/tradereport_",(("_" sv string dates) except "."),".csv") 0: "," 0:
    tradereport;


-1 (string .z.p),":Report complete. Exiting...";
exit 0
```

