AquaQ Analytics Limited

kdb+ Training

Presented by Matt Doherty

email:info@aquaq.co.uk

Course Outline

email:info@aquaq.co.uk

Course Outline

18. ETL
19. Idioms
20. TorQ

email:info@aquaq.co.uk

Introduction

email:info@aquaq.co.uk

KDB+

- **k** data**b**ase **plus**
- High performance database, managing real time and historical data within a single platform
- Technical Features:
  - Column orientated
  - Embedded query language - q
  - In-memory and on-disk data access
  - Optimized and configurable partitioning of on-disk data
  - 64-bit architecture with built-in multi-threaded support
  - Direct analysis on data
  - High speed time series analysis
  - Discrete and continuous joins
  - Small and portable (500kB vs 3.2GB min for Oracle)

email:info@aquaq.co.uk

KDB+

- Reference website: `http://code.kx.com`
- Reference card: `http://code.kx.com/q/ref/card/`
- Learn kdb+ in X minutes `https://learnxinyminutes.com/docs/kdb+/`

email:info@aquaq.co.uk

kdb+ features

- kdb is simple because it does exactly what you tell it to; no intermediate layers like query optimizers to fight with. this is for better and for worse. main takeaway: don't over-complicate it!
- the language is different from most others. APL roots, preference for short built-in-function names, internal function over-loading, user-defined functions, chaining commands, table operations
- language is very dense with little coding overhead. for better and for worse
- query language is particularly powerful and consistent
- right-to-left evaluation is unambiguous
- simple things are simple; hard things are possible
- It's mature (first released 2003, precursor k released in 1993)

KDB+ Installation

- http://kx.com/software-download.php
- Unzip the file and put the q_VV_yyyymmdd folder in C drive
- Going into "q_VV_yyyymmdd" then "w32" is the home directory of "q_VV_yyyymmdd".
- Changing the environment variable path to access the "q_VV_yyyym
  home directory means you can access q in any directory.
- Go to "start", right click on "computer", click "properties"
- Click "advanced system settings" then "environment variables"
- In "system variables", select "path" and click "edit" and add in ";C:\q_VV_yyyymmdd\w32\"
- In user variables, add QHOME as "C:\q_VV_yyyymmdd\"
- Open a command prompt and type "q" to start a q session

 email:info@aquaq.co.uk

KDB+ Installation

- In unix/mac, “QHOME” can be reset by “export QHOME=/<dir>”
- Download help.q from `http://code.kx.com/wsvn/code/kx/kdb+/d/help.q`

email:info@aquaq.co.uk

Basic Overview

email:info@aquaq.co.uk

Data Types

- kdb+ has 3 data constructs: atom, list, and dictionary
- Some examples:

```
q)/starts a comment
q) 1                   / atom - single data point
q) (1;2)               / list of atoms - full syntax
q) 1 2                 / list of atoms - shorthand syntax
q) (1;"a")             / list of atoms - full syntax since mixed types
q) (1 2;3 4)           / list of lists (here a square matrix)
q) (1 2)!(3 4)         / dictionary (key-value mapping atoms to atoms)
q) 1 2!3 4             / don't actually need brackets
q) 1 2!(3 4;5 6)       / dictionary (map atoms to lists)
q) "abcde"             / string - list of characters
q) `abcde              / symbol - one atom; internalized; little duplication
```

- Sufficient data types: `http://code.kx.com/wiki/Reference/Datatype`
- Try to remember the common type letter, type name, and type number as they have widespread use

Data Types

- How the console displays data types

```
q) 1j
1                    / missing: j is default integral type
q) 1.1f
1.1                  / missing: f is default floating point type
q) 1i
1i                   / not the default data type so shows type letter
q) 1.0f
1f                   / displays f since `1' would mean long
q)(1;2)
1 2                  / displays as shorthand
q)(1f;2f) / shorthand means all elements are the same type
1 2f                 / f means they are all float
q)1 2!(3 4;5 6)
1| 3 4               / dictionary key(s) | value(s)
2| 5 6
```

- type function

```
q)type 100i
-6h                  / atom so a negative number. what does the 'h' mean?
q)type 100 200i
6h                   / simple list so positive number
q)type (1;"a")
0h                   / mixed list so 0h
```

Data Types

- Type cast ($):

```
q)`int$1h       / name representation
1i
q)6h$1h         / number representation
1i
q)"i"$1h        / character representation
1i              / change from type short to type int
q)`int$3.3f
3i              / rounds when from type float to type int
q)`int$()
            `int$()         / return an empty list of type int
q)`$"AAPL,asdf" / build a more complicated symbol by string casting
```

- String parsing ($ with uppercase type letter):

```
q)"D"$"2016.01.01"
2016.01.01
q)"J"$"1"
1
q)"F"$"1"
1f
q)"S"$("hello")       / enumerated string type (symbol)
`hello
```


email:info@aquaq.co.uk

Data Types

- Built in temporal types:

```
q)d:2017.05.08                   / date
q)t:10:00:00.000                 / time
q)dt:2017.05.08D10:00:00.000  / and datetime (normal arithmetic applies)
```

- Implicit type conversion:

```
q) if[count 1 2 3;"do something"]  / count returns number of items
q) 1+1.0                           / long+float => float
q) 1*2                             / 2*long => long
```

- Each data type has nulls and infinities:

```
q)0N                      / default null long
q)0Nf                     / default null float
q)0Nd                     / default null date
q)0W                      / default infinity long
q)0Wf                     / default infinity float
q)0Wd                     / default infinity date
q)sum 0N 1 2              / most functions properly deal with nulls
q)min`long$()             / some conditions return infinity
```

Operations

- Basic arithmetic: + - * %
- Other calculus: min max avg mid ...
- Operator consistency across data structures

```
q) 1+1                / 1+ atom
q) 1+1 2              / 1+ list
q) 1+1 2!(3 4;5 6)    / 1+ dictionary
q) 1 2+3 4            / list + list
q) 1 2+3 4!5 6        / list + dictionary
q) (1 2!3 4)+2 3!5 6  / dictionary + dictionary

q) 1+(1;"a")          / fails - data type mismatch
q) 1 2+3 4 5          / can't add lists of different length
```

- Right to left execution (try 5 - 2 + 3)
  - Best to reorder your expression like 5+3-2
  - Or you can use brackets like (5-2)+3
- Assign variables using ':' instead of '='

email:info@aquaq.co.uk

Operations

- '=' and '∼' are used for comparison
- Comparison can be:

```
>=  / Greater than or equal to
>   / Greater than
<=  / Less than or equal to
<   / Less than
=   / Equal (can error out like 1=`a)
<>  / Not equal
~   / Boolean result showing if two structures match (never errors 1~`a)
```

- Assignments can be combined in a single operation:

```
q)a+:2                    / don't need to preinitialize a
q)a:a+2

q)l:1 2 3 4 5
q)l*:2
q)l
2 4 6 8 10
```

Operations

- Multi-assignment can be performed on one line:

```
q)c:1+a:2+b:3
q)a
5
q)b
3
q)c
6
```

- Calling functions Infix vs Postfix

```
q)1+2               / infix (recommended by Kx)
q)+[1;2]            / postfix
```

- Not much special about built-in functions

```
q)f:+
q)f[1;2]
3
q)1+2
q)1 f 2       / fails - only built-in functions can be called with infix
```

- You can chain commands together (within reason!)

```
q)(`long$0.1*count l)#l:til 100 / take top 10% of l
```

Console

- In training, we work on the console since nothing is hidden from you and it's possible to debug
- In practice, most people use a UI because the console is hard to enter multi-line statements etc
- The console has several modes:

```
q)                    / default 'q' language mode
q)\
/ single \ to escape into 'k' language mode - no q)
\                     / single \ to return to q mode
q){1+x}[`a]           / user defined function with an error
{1+x}                 / which function crashed
'type                 / with which error
+                     / expression that failed
1
`a
q))                   / multiple ) means you're in the debug prompt
q))\                  / single \ to move up one level
q)\\                  / double backslash exits kdb+
```


email:info@aquaq.co.uk

Console

- There are some very useful internal functions:

```
q)a:1                          / assign 1 to a
q)\v                           / return which variables are defined
q)system"v"                    / same as \v but you can use it in expr
q)\a                           / return which tables are defined
q)\t {exp x}each til 100       / expression runtime
q)\ts {exp x}each til 100      / expression runtime & memory
q)\t:100 {exp x}each til 100   / runtime of 100 evaluations
q)\c                           / return console height and width
q)\c 25 1000                   / set console height+width
```

- Scripts:

```
q)\l script.q         / load script in current folder with name script.q
                      / can also be located in $QHOME
                      / the .q tells kdb+ to use the q language to parse
q)system"l script.q"  / same as above but you can construct script.q
q)\l help.q           / load script help.q
```

- Statements

```
q)a:1;               / no output - last statement ends in ;
q)a:1;a              / multiple statements on one line
```

Console

- Control flow

```
/ if is as you might expect (; separates the condition and instructions)
q)if[1=1;a:"hi"]
q)a
"hi"
/ if-else uses $ (and unlike if, returns a value)
q)$[1=0;a:"hi";a:"bye"]
"bye"
q)a
"bye"
/ if-else can be extended to multiple clauses by adding args separated by ;
q)$[1=0;a:"hi";0=1;a:"bye";a:"hello again"]
"hello again"
```


email:info@aquaq.co.uk

Exercises: Atoms and Basic Operations

Please refer to kdbexercises.pdf

Complete chapter 1 on Atoms and Basic Operations.

email:info@aquaq.co.uk

List

- A collection containing 0 or more items, with or without a data type
- Declaration syntax:

```
q)l:(1;2;3;4)          / this syntax will work for any types
1 2 3 4
q)l:1 2 3 4            / for single typed lists there are shortcuts...
1 2 3 4
q)l:1 2 3 4f
1 2 3 4f
q)l:"hello world"
"hello world"
q)`IBM`AAPL`MSFT
`IBM`AAPL`MSFT
```

- Data type associated with non-negative number

List

- Basic arithmetical operations and logical operations work item by item on lists, as they would with atoms:

```
q) show ls: til 5
0 1 2 3 4
q) ls * 5
0 5 10 15 20
q)ls > 3
00001b
```

- Operations on a list are much faster than the same number of operations on individual atoms

```
q)\t 1+til 10000000
124
q)\t:10000000 1+1
874
```

- Mathematical functions such as min, max, avg ... act on all elements of the list - as would be expected:

```
q) avg ls
2f
q) max ls
4
```


email:info@aquaq.co.uk

List

email:info@aquaq.co.uk

List Types

- Simple list - contains atoms with same data type:

```
q)l1: (100i;200i;300i;400i)
q)l1
100 200 300 400i
q)type l1
6h
q)type each l1
-6 -6 -6 -6h
```

- Empty list - contains nothing:

```
q)()
q)type ()
0h
q)type `int$()          /empty list with specific type
6h
```


email:info@aquaq.co.uk

List Types

- Mixed list - contains atoms with different data types:

```
q)l2: (100i;200h;300j;400e)
q)l2
100i
200h
300
400e
q)type l2
0h
q)type each l2
-6 -5 -7h -8h
```

- Comparison of a simple list and a mixed list:

```
q)l1 ~ l2
0b                   /match (~)considers the different data types
q)l1 = l2
1111b                /the = operator does not consider data types
```


 email:info@aquaq.co.uk

List Types

- Singleton list - contains only one element:

```
q)enlist 100i
,100i
q)type enlist 100i
6h
q)type each enlist 100i
,-6h
```

- Nested list - depth level of 2 or more:

```
q)(1;2;(3;4))
1
2
3 4
q)type (1;2;(3;4))
0h
```


email:info@aquaq.co.uk

List Indexing

- Use index to access items in list:

```
q)l:1 2 3 4 5 6
q)l[0]
1
q)l[0 3 1]          /index using a list
1 4 2
```

- Index assignment:

```
q)l[2]:10
q)l
1 2 10 4 5 6
q)l[0]:1.1          /the assigned value must match the type of list
'type
q)a:(`a;2;3.15)
q)a[0]:"a"          /unless the list is a mixed list
q)a
"a"
2
3.15
q)a[0 1]:`a`b       /index assign using list
q)a
`a
`b
3.15
```

List Indexing

- Matrix - list with depth 2:

```
q)mm:(1 2 3;4 5 6;7 8 9)
q)mm
1 2 3
4 5 6
7 8 9
```

- Indexing at depth:

```
q)mm[0;]           /take all from first row
1 2 3
q)mm[;0]           /take first item from all rows
1 4 7
q)mm[1;1]          /take second item from second row
5
q)mm[2;10]         /returns null of proper type as column
0N                 /index is out of range
```


email:info@aquaq.co.uk

List Indexing

- Index assignment at depth:

```
q)mm[0;1]:12
q)mm
1 12 3
4 5  6
7 8  9
```


email:info@aquaq.co.uk

Basic List Operations

- Iteration (each)

```
q)show m:(1 2;3 4 5 6;7 8 9)
1 2 3
4 5 6
7 8 9
q)count m
3
q)count each m
2 4 3
```

- Join (,)

```
q)1,2                   /atom to atom
1 2
q)`a,`b`c`d             /atom to list
`a`b`c`d
q)1 2 3,4 5 6           /list to list
1 2 3 4 5 6
q)1 2, `a`b             /joining two lists of different types
1
2
`a
`b
```


email:info@aquaq.co.uk

Basic List Operations

- Drop (_)

```
q)2_1 2 3 4 5                /drop first 2
3 4 5
q)-2_1 2 3 4 5               /drop last 2
1 2 3
q)1 2 3 4 5 _ 2              /drop item with index 2
1 2 4 5
```

- Cut (_)

```
q)1 3 4 cut 1 2 3 4 5 6   /cut at index 1, 3 and 4
2 3
,4
5 6
```

Basic List Operations

- Take (#)

```
q)2#1 2 3 4 5                  /take first 2
1 2
q)-2#1 2 3 4 5                 /take last 2
4 5
q)10#1 2 3 4 5                 /repeat take
1 2 3 4 5 1 2 3 4 5
q)2 3#1 2 3 4 5                /2x3 matrix
1 2 3
4 5 1
q)1#1                          /works on atoms
,1                             /short way to create singleton lists
```

- Sublist

```
q)2 sublist 1 2 3 4 5          /take first 2
1 2
q)-2 sublist 1 2 3 4 5         /take last 2
4 5
q)10 sublist 1 2 3 4 5         /take only what's available
1 2 3 4 5
q)2 3 sublist 1 2 3 4 5        /takes 3 items starting from position 2
3 4 5
```

- Find (?)

Basic List Operations

- As you will see with the ? operator, it has multiple different uses - this is common in q (overloading).
  - If the left argument is a list and the right argument an atom of the same type, i.e. list ? atom **finds** the first occurance of the atom in the list.

```
q)lst:8 1 9 5 4 6 6 1 8 5;lst
8 1 9 5 4 6 6 1 8 5
q)lst?8
0
q)1 2 3 4 5?7           /not found
5                       /returns 1+count list
q)1 2 2 3 4?2           /only returns the first occurrence
1
```

  - A number as the left argument to the ? operator and a list as the right argument, number ? list would randomly select number amount of elements from list.

```
q)5?lst
6 6 4 1 5
```


 email:info@aquaq.co.uk

Basic List Operations

- Random (?) - useful for creating sample data

```
q)5?1 2 3 4 5           /pick 5 random items from the list
1 3 2 2 4
q)5?10                  /pick 5 random items less than 10
8 5 5 9 2
q)5?`2                  /pick 5 random symbols with length 2
`ab`hg`ij`fr`dc
q)5?" "                 /pick 5 random characters
"akdlm"
q)-5?10                 /pick 5 distinct random items less than 10
1 6 2 9 4
```


email:info@aquaq.co.uk

Basic List Operations

- Count

```
q)count 1 2 3 4           /count number of elements in list
4
q)count enlist 1 2 3 4    /counts the first dimension of the enlisted
1                         /structure
q)count (1 2 3;4 5 6)     /2 lists in a nested list
2
q)count each (1 2 3;4 5 6) /count each list in a nested list
3 3
q)count ()                /works on an empty list
0
q)count 5                 /works on an atom
1
```


email:info@aquaq.co.uk

Basic List Operations

- First, Last

```
q)first 42                /operates on atoms and lists
42
q)ls
3 7 8 2 1 4 2 8 0 5
q)first ls
3
q)last ls
5
q)ls2:enlist(10)
q)ls2
,10
q)first ls2               /acts as a dual to `enlist'
10
q)nls:(1 2; 4; 5 6 7; 8 9; 0 10 11 12 13)
q)first each nlst         /can be used on each row of a nested list
1 4 5 8 0
```


 email:info@aquaq.co.uk

Basic List Operations

- raze

```
q)type (1 2 3;4 5 6)
0h
q)raze (1 2 3;4 5 6)     /raze removes one level of structure
1 2 3 4 5 6
q)type raze (1 2 3;4 5 6)
7h
q)raze a:(1 2;4 5;(8 9;10 11))
1
2
4
5
8 9
10 11
q)0N!raze a /only one level removed
(1;2;4;5;8 9;10 11)
1
2
4
5
8 9
10 11
q)(raze/) a /can be combined with over adverb to produce single list
1 2 4 5 8 9 10 11
```


email:info@aquaq.co.uk

Basic List Operations

- Match ($\sim$)

```
q)list1: 1 2 3 4
q)list2: 1 2 3 4
q)list3: 1 2 3 5      /different elements in list3

q)list1~list2         /returns true if all components match
1b
q)list1~list3         /returns false if some or all components mismatch
0b
q)list2=list3         /checks each component of a structure and returns
1110b                 /a list of boolean values

/~ can only return true if = returns true across an entire object!

q)(distinct list1 = list2) ~ enlist (list1 ~ list2)
1b
```

- In the last example, the = comparison is distinctly true and matches to the $\sim$ comparison
- $\sim$ considers data type, = does not

email:info@aquaq.co.uk

Basic List Operations

- And a few more that will be particularly useful/relevant later in tables
- where

```
/ If passed a boolean list, this returns the indices of the 1's
/ Thus where is often used after a logical test
/ Where is also a crucial part of q-sql statements

q)where 0 1 1 0 1
1 2 4
q)x:1 5 6 8 11 17 20 21
q)where 0 = x mod 2         / indices of even numbers
2 3 6
q)x where 0 = x mod 2       / select even numbers from list
6 8 20
```


email:info@aquaq.co.uk

Basic List Operations

- like

```
/ like is the q pattern-matching primitive

q)X like Y              / X can be sym or char or string
/"?" matches an arbitrary character in the pattern
/"*" matches an arbitrary sequence of characters in the pattern
/"[]" are used in pairs to list alternatives
q)a:("roam";"rome")
q)a like "r?me"
01b
q)a like "ro*"
11b
q)a like "ro[ab]?"
10b
q)a like "ro[^ab]?"
01b
q)"a[c" like "a[\\[]c"
1b
```


 email:info@aquaq.co.uk

Basic List Operations

- ss and ssr

```
/ function ss finds positions of a substring within a string
q)s:"toronto ontario"
q)s ss "ont"
3 8

/ function ssr does search and replace on a string
q)s:"toronto ontario"
q)ssr[s;"ont";"x"]
"torxo xario"
```

- fills

```
/function that is used to forward fill a list containing nulls

q)fills 0N 2 3 0N 0N 7 0N
0N 2 3 3 3 7 7

/ very useful for filling blanks in tables sorted by time
```


email:info@aquaq.co.uk

Basic List Operations

- xbar

```
/ The xbar verb rounds its right argument down to the nearest
/ multiple of the integer left argument

/ This is very useful for bucketing data by time

q)3 xbar til 16
0 0 0 3 3 3 6 6 6 9 9 9 12 12 12 15
q)5 xbar 11:00 + 0 2 3 5 7 11 13
11:00 11:00 11:00 11:05 11:05 11:10 11:10
```


email:info@aquaq.co.uk

More Built-in Operations

- distinct - returns the distinct items within a group of entities
- except - excludes the specified item from a list or dictionary
- flip - takes a simple list, column, dictionary, or table and transposes it
- group - applied to a list, returns a dictionary of positions for each distinct element
- in - returns a boolean result on whether a specified item is in a list
- next - shifts each item in a list one position to the left. nulls are padded on the RHS
- prev - shifts each item in a list one position to the right. nulls are padded on the LHS
- xnext/xprev - as next and prev, but allows specification of the number of positions to shift

email:info@aquaq.co.uk

More Built-in Operations

- reverse - reverses the order of items in a list, dictionary, or table
- string - this can be applied to any data type and the result will be a list of characters forming a string
- til - returns a list of integers from 0 to $n-1$
- value - this can be applied to a dictionary or a table to get the range of items contained, or give a value result to a string expression. It can also be used to dereference globals:

```
f:{x+1};value[`f]
```

- within - returns a boolean result depending on whether the specified item is within the bounds specified.

email:info@aquaq.co.uk

”Real” example

- Project Euler problem 1:
- Find the sum of all the multiples of 3 or 5 below 1000

```
q)l:til 1000
q)d1:l where 0=l mod 3
q)d2:l where 0=l mod 5
q)sum distinct d1,d2
233168
```

- This can actually be done much more concisely, which we’ll come back to later...

email:info@aquaq.co.uk

Apply and Amend

The operators . and @ are used for apply and amend.

- Index, apply/amend at top level @
  - Diadic @ indexes at top level

```
q)n1: ((1 2 3; 4 5); (6 7; 8 9 10 11); (12; 13 14 15; 16 17))
q)n1@0 1 / choose items at indices 0 and 1
1 2 3 4 5
6 7   8 9 10 11
```

  - Apply applies the arguments to monadic functions

```
q)sum 1 2 3    / preferred notation (generally)
q)sum[1 2 3]   / alternate functional notation
q)sum@1 2 3    / alternate apply notation
```

Apply and Amend

- 3, 4 arguments @ apply/amend at top level

```
q)n2: til 10
q) / apply `+' with second arg 10 to the elements at indices 1 2 3
q)@[n2;1 2 3;+;10]
0 11 12 13 4 5 6 7 8 9
q) / to amend the values (pass by ref) send sym i.e. `a2:
@[`n2;4 5;*;2]
`n2
q)n2
0 1 2 3 8 10 6 7 8 9
q)
q) / apply function {x*10} to the elements of n2 at indices 2 3
q)@[n2;2 3;{x*10}]
0 1 20 30 4 5 6 7 8 9
```

- Index, apply/amend at depth .
  - Diadic . index at depth

```
q)n1: ((1 2 3; 4 5); (6 7; 8 9 10 11); (12; 13 14 15; 16 17))
q)n1 . 0 1 /take element at 0, then the element at index 1 of this
4 5
q)n1 . (0 1;1) /take index 1 elements of values at indices 0 and 1
4 5
8 9 10 11
```


email:info@aquaq.co.uk

Apply and Amend

- Apply at depth applies the arguments to multivalent functions

```
q)f:+
q)f[1;2]       / preferred notation (generally)
q)f . 1 2      / alternate apply at depth notation
q)f[1 2;3]     / preferred with list
q)f . (1 2;3)  / alternate apply at depth with list
```

- 3, 4 arguments . apply/amend at depth

```
q) /apply `%' with second arg 10 to the elements at indices (0 1;1)
q).[n1;(0 1; 1);%;10]
(1 2 3;0.4 0.5)
(6 7;0.8 0.9 1 1.1)
(12;13 14 15;16 17)
q)
q) /apply function {x+100} to the elements at indices (0 1;1)
q).[n1;(0 1;1);{x+100}]
(1 2 3;104 105)
(6 7;108 109 110 111)
(12;13 14 15;16 17)
q)
```


email:info@aquaq.co.uk

Common Errors

- 'length
  Incompatible length:

```
q)1 2+3 4 5
'length
```

- 'type
  Wrong type:

```
q)`a + 10
'type
q)ls: 1 2 3,`b`c
q)avg ls
'type
```

- 'rank
  Invalid rank or valence:

```
q) /e.g. trying to supply 3 args to a diadic fn
q) f:{x*y}
q) f[3;4;5]
`rank
```


email:info@aquaq.co.uk

Exercises: Lists

Please refer to kdbexercises.pdf

Complete chapter 2 on Lists.

email:info@aquaq.co.uk

Dictionaries and Tables

Dictionaries

- A dictionary is a mapping between a domain list (key) and range list (value)
- A foundation of a table (table is a list of dictionaries)
- Dictionary lookup:

```
q)dy:`a`b`c!(1 2 3;`a`b`c;7 8 9)
q)dy
a| 1 2 3
b| a b c
c| 7 8 9
q)dy[`a]
1 2 3
```

- Joining dictionaries:

```
q)dy1:(`a`b`c)!(7 8 9;`d`e`f;101b)
q)dy1,:`c`d!8 7                       /join in place
q)dy1
a| 7 8 9
b| `d`e`f
c| 8                                  /existing key is overwritten
d| 7                                  /new key is added
```


email:info@aquaq.co.uk

Dictionaries

- Lookup assignment:

```
q)dy2:`a`b`c!(1 2 3;4 5 6;7 8 9)
q)dy2[`c]:9 3 6          /amend existing key
q)dy2
a| 1 2 3
b| 4 5 6
c| 9 3 6
q)dy2[`d]:8 8 8          /add new key
a| 1 2 3
b| 4 5 6
c| 9 3 6
d| 8 8 8
q)dy2[`a;0]:3f           /type safe
'type
```

- Operations on dictionaries.

email:info@aquaq.co.uk

Dictionaries

- Arithmetic operators act on the range lists of dictionaries for each element they have in common. Other domain elements are present but remain unchanged.

```
q)d1:`alpha`bravo`charlie`delta`echo`foxtrot!10 15 3 21 6 30
q)d2: `alpha`charlie`bravo`foxtrot!10 8 6 30
q)
q)d2*2
alpha  | 20
charlie| 16
bravo  | 12
foxtrot| 60
q)
q)d1+d2              /result shows all the distinct domains,
alpha  | 20          /and only adds those present in both
bravo  | 21
charlie| 11
delta  | 21
echo   | 6
foxtrot| 60
```


email:info@aquaq.co.uk

Dictionaries

- Logical operators will also match on dictionary ranges according to keys:

```
q)d1=d2
alpha  | 1
bravo  | 0
charlie| 0
delta  | 0
echo   | 0
foxtrot| 1
```

- We can also use other mathematical operations:

```
q)dy2:(`a`b`c)!(1 2 3;4 5 6;7 8 9)
q)avg dy2                /average on whole dictionary by index
4 5 6f
q)avg each dic           /average on each value of dictionary
a| 2
b| 5
c| 8
```


 email:info@aquaq.co.uk

Exercises: Dictionaries

Please refer to kdbexercises.pdf

Complete chapter 3 on Dictionaries.

email:info@aquaq.co.uk

Relationship between Dictionary and Table

- Relationship between a dictionary and a table

```
q)show dy2:`a`b`c!(1 2 3;4 5 6;7 8 9)
a| 1 2 3
b| 4 5 6
c| 7 8 9
q)show tab:flip dy2
a b c
-----
1 4 7
2 5 8
3 6 9
q)type tab: flip dy2
98h                      /unkeyed table
q)show tab2:([]a:1 2 3;b:4 5 6;c:7 8 9)
a b c
-----
1 4 7
2 5 8
3 6 9
q)tab2~tab               /both are the same
1b
q)dy2~flip tab2          /table can be flipped back to dictionary
1b
```

Relationship between Dictionary and Table

- The value in a dictionary must be a list in order to flip to a table

```
q)show dy4:`a`b`c!1 2 3
a| 1
b| 2
c| 3
q)type each dy4
a| -7                    /values are atoms
b| -7
c| -7
q)tab:flip dy4
'rank
q)show tab:enlist dy4    /put the dictionary elements into a list
a b c                    /as table is a list of dictionaries
-----
1 2 3
q)flip enlist each dy4    /convert each atom to singleton list then flip
a b c
-----
1 2 3
q)tab~flip enlist each dy4
1b
```

-

email:info@aquaq.co.uk

Tables

- A collection of named columns or a list of dictionaries
- Simple table

```
q)([]a:1 2 3;b:`a`b`c;c:7 8 9)
a b c
-----
1 a 7
2 b 8
3 c 9
```

- Keyed table

```
q)([a:1 2 3]b:`a`b`c;c:7 8 9)
a| b c                                  /dictionary of tables
-| ---
1| a 7
2| b 8
3| c 9
```

- Empty table

```
q)([]a:`int$();b:();c:())
a b c
-----
```


email:info@aquaq.co.uk

Tables

- Use 'meta' to retrieve table's attributes
- Empty table meta type

```
q)meta ([]a:();b:();c:())
c| t f a
-| -----
a|
b|
c|            /empty meta table as there is no information held
```

- Simple table with attributes

```
q)meta ([]a:`s#1 2 3;b:`g#`a`b`c;c:`u#"abc")
c| t f a
-| -----
a| j   s
b| s   g
c| c   u
```

- 'c' gives the column names
- 't' gives the column types
- 'f' gives the foreign keys
- 'a' gives the attributes

email:info@aquaq.co.uk

Tables

- There are four possible attributes:

Attribute Name	Description
Sorted (‘s#)	The items in the list are in sorted order
Unique (‘u#)	No duplicates within a list
Group (‘g#)	Create a dictionary that maps each occurrences to their position
Parted (‘p#)	Create a dictionary that maps to the first occurrence

- For further information, see the later section on Attributes in this document, and also: `http://code.kx.com/wiki/JB:QforMortals/tables#Attributes`
- Meta table with nested lists:

```
q)meta ([]a:((1;2);(3;4));b:((2;5);(7;9)))
c| t f a
-| -----
a| J
b| J                   /the capital letter correlates to a nested list
                       /of long types
```


email:info@aquaq.co.uk

Tables

- Foreign key is a field in one table that uniquely identifies a row of another table

```
q)tabid:([id:5625 5626 5627 5628];sym:`a`b`c`d)
q)tab:([]id:`tabid$5625 5626 5627 5628;price:101.1 102.3 103.5 104.6)
q)meta tab
c    | t f     a
-----| ---------
id   | j tabid
price| f
q)select id,id.sym,price from tab
id   sym price
--------------
5625 a   101.1
5626 b   102.3
5627 c   103.5
5628 d   104.6
```


email:info@aquaq.co.uk

Tables

- Virtual column i within a table represents the index of each row (in this case i has its own operation, it cannot be assigned as a column)

```
q)select i from tab
x
-
0
1
2
3
q)exec i from tab
0 1 2 3
```


email:info@aquaq.co.uk

Tables

- Extract rows and columns:

```
q)t:([]a:1 2 3;b:4 5 6;c:7 8 9)
q)
q)t[2]                              /take all columns from third row
a| 3
b| 6
c| 9
q)t[`a]                             /take column `a from all rows
1 2 3
q)t[`a`b]
1 4
2 5
3 6
```

- Reverse lookup:

```
q)t?`a`b`c!3 6 9                    /find one row using dictionary
2
q)t?(3 6 9)                         /find one row using list
2
q)t?((2 5 8);(3 6 9))               /find 2 rows using list
1 2
q)t?([]a:1 2;b:4 5;c:7 8)           /find using table
0 1
```

Tables

- Operations on tables
- Wtih **keyed tables**, then we can use arithmetic (as shown previously for dictionaries) and the operations will match on the keys.
- The result contains all keys present, and the ranges remain unchanged if not common to both.

```
q)kt1:([ks: `a`b`c] v1: 1 2 3; v2: 10 11 12)
q)kt2:([ks:`b`c`e] v1: 20 30 40; v2: 9 8 7)
q)
q)kt1+kt2
ks| v1 v2
--| -----
a | 1  10
b | 22 20
c | 33 20
e | 40 7
```


email:info@aquaq.co.uk

Tables

- With **unkeyed tables**, we can only use arithmetic if the table consists of numeric fields and row counts match.

```
q)t1:([]a: 1 2 3; b: 11 12 13f)
q)t2:([]a:4 5 6; b: 10 11 12)
q)t3:([]a:20 21 22 23; b:100 101 102 103)
q)t1+t2
a b
----
5 21
7 23
9 25
q)t1+t3
'length
```


email:info@aquaq.co.uk

Exercises: Tables

Please refer to kdbexercises.pdf

Complete chapter 4 on tables.

email:info@aquaq.co.uk

Assignment

email:info@aquaq.co.uk

Assignment

- Here are some examples of assign:

```
q)a:1                    / assign 1 to a
q)a+:1                   / add 1 to a. a doesn't need to be pre-defined
q)l:1 2 3
q)l[1]:4                 / indexing at depth
q)m:3 3#til 9
q)m[1;1]:10              / replace middle of m with 10
q)m[;0]:10 11 12         / replace first column with 10 11 12
```

- You can also assign globals using 'set':

```
q)`zz set 1
q)zz
1
```

Assignment

- Assignment tries to 'fit' the new data into the current structure, which can make operators act differently in other contexts:

```
q)a:`s#til 10
q)a,10                        / join 10 to a
0 1 2 3 4 5 6 7 8 9 10        / loses `s# attribute
q)show a,:10                  / join 10 to a and assign
`s#0 1 2 3 4 5 6 7 8 9 10     / keeps `s# attribute
q)z,1                         / fails if z isn't defined
q)z,:1                        / z doesn't need to be predefined
q)t:([]a:1 2 3;b:4 5 6)
q)t,([]b:7 8)                 / can't , because missing column a
'mismatch
q)t,:([]b:7 8)                / but can ,: because assign fits the new
q)t                           / data into the existing structure
a b
---
1 4
2 5
3 6
  7
  8
```


 email:info@aquaq.co.uk

Assignment

- Local variables are initialized to () before assignment

```
q){x+1;b:10}[`a]
{x+1;b:10}
'type
+
`a
1
q))b
()
```


 email:info@aquaq.co.uk

Functions

email:info@aquaq.co.uk

User Defined Functions

- Basic template of a function:

```
q)f:{ [arg1] arg1+2}
```

: assigns the function definition to variable **f**

The text within **{ }** gives the function definition

The text within [ ] specifies the function arguments (separated by **;**)

If no [ ] is present, Q tries to infer arguments x, y, z (exclusively) from the definition (ie: {x+1} kdb interprets as requiring a single x argument; {y+1} sets both x and y are arguments; {a+1} will fail unless a is a global)

: sets the return value and interrupts the function

If the last statement doesn't have a **;** to suppress output; it becomes the return value

email:info@aquaq.co.uk

User Defined Functions

If there's no explicit or implicit return, :: (the global null) is returned

- Local scope hide the value of globals:

```
a:1
{a:10;a+1}[]      / returns 11 - but doesn't change a
```

- To set a global inside a function, use the set function:

```
{`a set 10}[]
a                 / returns 10
```

- Niladic function - no arguments

```
q)f:{1+1}
q)f[]
2
```

- Monadic function execution: - one argument

```
q)f:{x+1}
q)f[2]
q)f 2
q)f @ 2
```

User Defined Functions

- Dyadic function execution: - two arguments

```
q)f:{x+y}
q)f[1;2]
q)f.1 2
```

- User defined functions are not 'special'

```
q)f:+
q)+[1;2]
q)f[1;2]
q)1+2
q)1 f 2
'type
\- this syntax only works for dyadic functions in the
\- .q namespace
```

- Triadic function - three arguments

```
q)f:{x+y+z}
q)f[6;7;8]
21
```

User Defined Functions

- Multivalent function - more than three arguments

```
q)f:{[a;b;c;d;e;f] a+b+c+d+e+f}
q)f[1;2;3;4;5;6]
21
```

- Variable overload - maximum 8 arguments

```
q)f:{[a;b;c;d;e;f;g;h;i;j] a+b+c+d+e+f+g+h+i+j}
'params
/- error as variables exceed limit
```

- We can get around this using a single list or dictionary argument:

```
q)f:{[d] sum d}
q)f 1 2 3 4 5 6 7 8 9 10
```

User Defined Functions

- Alternatively, we can use some required arguments and an optional list/dict argument:

```
q)f:{[a;b;d] a+b+$[count d;sum d;0]}
q)f[1;2;()]
3
q)f[1;2;3]
6
```


email:info@aquaq.co.uk

User Defined Functions

- Type of a function

```
q)type {[x;y;z] d:x+y+z+12;.k.k:1;10}
100h
```

- Structure of a function:

```
q)get {[x;y;z] d:x+y+z+12;.k.k:1;10}
0xa07a41794178410316020d0b04810002a10004     /byte representation
`x`y`z                                       /argument names
,`d                                          /local variable
``.k.k                                       /global variable
12                                           /constant
10                                           /constant
"{[x;y;z] d:x+ y +z+12;.k.k:1;10}"           /string form
q)get[{[x;y;z] d:x+y+z+12;.k.k:1;10}][1]
`x`y`z          /get first element from the list - function arguments
```


email:info@aquaq.co.uk

User Defined Functions

- Function within a function

```
q)f:{x+1}
q)g:{f[x]*2}
q)g[2]
6
```

- In - line functions:

```
q)z:{f:{x+1};f[x]+2}   /defining a function within a function
q)z[3]
6
q)g:{{x+1}[x]+2}       /same result but without assigning to a variable
q)g[3]
6
```


email:info@aquaq.co.uk

User Defined Functions

- A **projection** of a function allows us to set one or more of the variables:

```
q)/- create simple diadic function
q)f:{x+y}
q)/- create a projection where x is fixed
q)g:f[2]
q)g
{x+y}[2]
q)g[10]          /g is equivalent to g:{2+y}
12
```

- A projection retains its definition if the original function is redefined:

```
q)f:{x*y*z}
q) /- create the original projection
q)h:f[10]
q) /-change the original function
q)f:{x+2*y*5*z}
q)h
{x*y*z}[10]
q)/- the projection doesn't change
```


email:info@aquaq.co.uk

User Defined Functions

- A projection of a projection is the same as a projection of the original function:

```
q)f:{x*y*z}
q)/- define h as a projection of f with x = 10
q)h:f[10]
q)h
{x*y*z}[10]
q)/-define n as a projection of h
q)n:h[;5]
q)/-this is equal to a projection of f; i.e. f[10; ;5]
q)n
{x*y*z}[10][;5]
q)n[2]
100
```


email:info@aquaq.co.uk

Exercises: Functions

Please refer to kdbexercises.pdf

Complete chapter 5 on Functions.

email:info@aquaq.co.uk

Scripts

email:info@aquaq.co.uk

Scripts

- A collection of q code saved in a text file
- File usually ends with .q extension
- 2 ways to load script:
  1. In a q session:
     ```
     q)\l script.q
     ```
  2. From the command prompt:
     ```
     > q script.q
     ```
- To suppress output within a script:
  ```
  f:{x+1};          /suppress output with ;
  f[2];
  f[5]*2;
  ```
- To force output within a script:
  ```
  f:{x+1};
  show f[2];       /using show function
  ON!f[5]*2;       /using ON! function
  ```

Scripts

- 'show' formats an object to plain text and writes it to standard output
- '0N!' returns the object that is passed into it

```
q)1+0N!2
2               /from the 0N!
3               /result of the expression
```

- Single line comment

```
a:1+2           /use "/" to comment
a
```

- Multi-line comment

```
/
Start with "/"
End with "\"
\
a:1+2
a
```


email:info@aquaq.co.uk

Scripts

- Multi-line statement

```
/When writing a script in a text editor, any function that carries
/onto the next line will need a space before writing the rest
/of the function

tab:([]sym:`symbol$();price:`float$();
 size:`int$();date:`date$())
|
put a space before continuing
```

- Note that this includes multi-line functions.

```
This won't compile, there is no white space
f:{[x]
x+1
}

This will compile
f:{[x]
          x+1
          }
```


email:info@aquaq.co.uk

Scripts

- Script error example:

```
tab:([]time:`time$();sym:`symbol$();
price:`float$())                  /continuation should start with space

`tab insert (12:30:30.000;`APPL;123.3)

 select from tab where sym=`APPL  /no need for a continuation space

/update thing                     /accidentally multi comment an update
update price+1 from tab           /query
\

sym:`AAPL;`YHOO;`MSFT             /a list without brackets
                                  /sym is assigned only to `AAPL

select price / 2 from tab         /standard division sign indicates
                                  /comment

show tab;                         \use wrong sign (\ instead of /)
                                  /to comment
delete time from tab              /multiline comment doesn't end \

exit 0
```


 email:info@aquaq.co.uk

Scripts

- Command line arguments can be used to supply information to a script before it is executed.
- This can be achieved by supplying arguments to a q command line:

```
C:\Users>q -key value
```

- This input can be brought into the q session by using .z.x, which returns the command line arguments as a list of strings:

```
q).z.x
"-key"
"value"
```

- Applying .Q.opt to the string list given by .z.x allows the output to be parsed as a dictionary.

```
q).Q.opt[.z.x]
key| "value"
```

Scripts

- Arguments prefixed with a '-' are parsed as keys, with the arguments following (delimited by spaces) being parsed as string values:

```
C:\Users>q -key1 value1 value2 -key2 value3 value 4
q).Q.opt[.z.x]
key1| ("value1";"value2")
key2| ("value3";"value";,"4")
```

- Depending on user input, a script can then have different applications (configuration files can also be used to achieve this).

email:info@aquaq.co.uk

Scripts

- You can supply defaults using .Q.def. This will also cast the argument to the correct type

```
$ q -abc 123 -xyz 321

q).Q.opt .z.x
abc| "123"
xyz| "321"

q).Q.def[`abc`xyz`efg!(1;2.;`a)].Q.opt .z.x
abc| 123
xyz| 321f
efg| `a
```


email:info@aquaq.co.uk

Scripts

- If you have a script *myscript.q*:

```
options:.Q.opt[.z.x];
if["date"~options[`print][0]; variable: .z.d];
if["time"~options[`print][0]; variable: .z.t];

variable
```

- This script will then decide whether to print the date or time based on a users input:

```
C:\Users>q myscript.q -print time
```

- The string values can also be cast into specific q datatypes for use in calculations or queries.
- .z.f can also be used to return the script name, supplied on the command line, as a symbol. This can be useful when logging information needs to be kept.

email:info@aquaq.co.uk

Exercises: Scripts

Please refer to kdbexercises.pdf

Complete section 6.1 on Scripts.

email:info@aquaq.co.uk

Operations on Tables

email:info@aquaq.co.uk

Select and Exec

- The basic template of a **select** query:

```
select <columns> by <groups> from <table> where <condition>
```

- The only required parameter in a select query is the ‘from’ condition. The others are all optional.
- A basic example to select required columns from a table using the template:

```
select size by dp from tab where size = 50
```

- The parts of the select query are evaluated in the following order:
  1. from
  2. where
  3. by
  4. select

 email:info@aquaq.co.uk

Select and Exec

- There are four ways to limit the return results from a select:
  1. Return the first n results:
     ```
     select[n] ...
     ```
  2. Return n results starting from position m:
     ```
     select[m n] ...
     ```
  3. Return the results by the specified order:
     ```
     select[order] ...
     ```
  4. Return the first n results by the specified order:
     ```
     select[n;order] ...
     ```

Select and Exec

- **Exec** works like a select statement, and has the same general template:

```
exec <columns> by <groups> from <table> where <condition>
```

- The difference is that an exec statement doesn't return a table.
- If the result is a single column of values, then it is returned as a list:

```
q) a: ([] c1: 1 2 3; c2: `a`b`c; c3: 8 9 20)
q) exec c2 from a
`a`b`c
```

- If the result is made up of values from multiple columns, the result is a dictionary:

```
q) exec c1,c2 from a
c1| 1 2 3
c2| a b c
```

Select and Exec

- We can return the first or last n results using take (#)

```
q) 2#exec c2 from a
`a`b
q) -2#exec c2 from a
`b`c
```

- For both **select** and **exec**, we can use the 'distinct' keyword to limit the results returned:

```
q)b:([] c1: `a`b`c`d`a; c2: 1 2 3 3 5; c3: 100 101 102 103 104)
q)select distinct c1 from b
c1
--
a
b
c
d
q)exec distinct c1 from b
`a`b`c`d
q)exec distinct c1, distinct c2 from b
c1| a b c d
c2| 1 2 3 5
/- note that for exec, distinct can be applied to multiple columns
```


 email:info@aquaq.co.uk

Update

- The basic template of and update query:

```
update <columns> by <groups> from <table> where <condition>
```

- Refer to the tablename to apply a modification.

```
/- to show the result of an update:
update price*2 from tab

/- to update the table in place:
update price*2 from `tab
```

- Can add a new column to the existing table:

```
/- add the new cols dp and ns to tab
update dp:price*2,ns:100+size from `tab
```

- Can update a subset of rows by applying a where condition:

```
update dp: price*2, ns:100+size from `tab where size =50
```

- Can update a column by applying a group-by clause:

```
update ap:avg price by size from tab
```


email:info@aquaq.co.uk

Delete

- To delete columns from a table:

```
delete <column> from <table>
```

- To delete rows from a table:

```
delete from <table> where <condition>
```

- The table has to be referred by name in order to modify the existing table:

```
q)tab:([]a:1 2;b:4 5)
q)delete a from tab
q)tab
a b
---
1 4
2 5
q)delete a from `tab
q)tab
b
-
4
5
```


email:info@aquaq.co.uk

Delete

- The delete template can also be applied to the workspace in order to delete any unwanted variables from your current session permanently (by ‘). The workspace can be accessed by using ‘’.’ :

```
q)\v                         /listing the variables in the workspace
`s#`tab`tab2
q)delete tab from `.
`.
q)\v                         /checking that tab is deleted
,`tab2
```

- If the workspace is referred to by a name, in this case ‘’aa’ then the variables in this workspace can also be deleted:

```
q).aa.a:1
q).aa.b:2
q)delete b from `.aa
q).aa
  | ::
a| 1
```

Insert

- To insert new data to the existing table, use the **insert** function:

```
`<table> insert <records>
```

- Must reference the tablename for the insert to work.
- The record is a list that matches the table column

```
q)tab:([]a:`a`b;b:1 2)
q)`tab insert (`b`c;3 4)
2 3
```

- The data types of the columns must correspond to the new inserted data types otherwise an error will occur

```
q)`tab insert (`d;`five)
'type
```

- Another common error is to refer to the table without a backtick resulting in a type error

Insert

- Insert returns the row indices that were added. If the table you are adding doesn't have all the columns, defaults are inferred:

```
q)`tab insert ([]a:`c`d)
4 5
q)tab
a b
---
a 1
b 2
c 5
d 6
c
d
```

- If the table is keyed, the new record can't match the existing keys:

```
q)tab:([a:`a`b];b:1 2)
q)`tab insert (`b`c;3 4)
'insert
```

  email:info@aquaq.co.uk

Insert

- As insert is dyadic and is also a verb, it can be written in many ways to obtain the same result:

```
insert[`<table>;<records>]
/- e.g. insert[`tab;(`d;6)]
```

```
insert[`<table>] (<records>)
/- e.g. insert[`tab] (`d;6)
```

- You can insert a record instead of lists of row values:

```
`<table> insert `<columns>!(<records>)
/- e.g. `tab insert `a`b!(`d;6)
```

- You can use this format to bulk insert records:

```
t1: ([] a:`c`d; b: 5 6)
e.g. `tab insert `t1
```

Upsert

- The basic template to **upsert** data is:

```
`<table> upsert <records>
```

- Records are in table format
- The table has to be referred by name in order to modify the existing table
- For an **unkeyed table**, upsert will just add a new row to the existing table - similar to 'insert'

```
q)tab:([]a:`a`b;b:1 2)
q)tab upsert ([]a:`b`c;b:3 4)
a b
---
a 1
b 2
b 3
c 4
```


email:info@aquaq.co.uk

Upsert

- Similar errors to the insert function can occur, with for example 'type':

```
q)tab upsert ([]a:'bc';b:3 4)
'type
```

- Similar to insert, the structure of the upsert template can vary also:

```
upsert[`<table>;<records>]
e.g. upsert[`tab;([]a:`b`c;b:3 4)]
`tab
```

```
upsert[`<table>] (<records>)
e.g. upsert[`tab] ([]a:`b`c;b:3 4)
`tab
```


email:info@aquaq.co.uk

Upsert

- For a **keyed table**, upsert will amend the existing table if the record matches the existing keys of the table.
  If the record doesn't match the existing keys, it will just add new rows to the existing table.

```
q)tab:([a:`a`b];b:1 2)
q)tab upsert ([a:`b`c];b:3 4)
a| b
-| -
a| 1
b| 3
c| 4
```

- Doesn't need to match the column order

```
q)tab:([a:`a`b];b:1 2;c:4 5;d:6 7)
q)tab upsert ([a:`a`b];d:8 9;c:1 2)
a| b c d
-| -----
a| 1 1 8
b| 2 2 9
```


email:info@aquaq.co.uk

Attributes and Queries

- Applying attributes to tables will improve the querying speed
- Commonly applied on frequently used columns, and columns which contain many different repeated values
- The columns that normally contain attributes are symbols, time and date
- Example:

```
q)\l fakedb.q
q)makedb[1000000;500000]
q)
q)quotes:update `#sym from quotes                    /without attribute
q)gquotes:update `g#sym from quotes                  /group attribute
q)pquotes:update `p#sym from `sym xasc quotes        /parted attribute
q)
q)\ts do[1000;select from quotes where sym=`IBM]
7976 12583456                                        /roughly 8 secs
q)\ts do[1000;select from gquotes where sym=`IBM]
4083 10486304                                        /roughly 4 secs
q)\ts do[1000;select from pquotes where sym=`IBM]
772 10486304                                         /less than 1 sec
```


 email:info@aquaq.co.uk

Attributes and Queries

- Order of selection in the where clause will effect the time of execution

```
q)\l fakedb.q
q)makedb[1000000;500000]
q)
q)\ts do[1000;select from trades where sym=`DELL,time<01:00:00.000]
3612 17044496                                        /roughly 3.6 secs
q)\ts do[1000;select from trades where time<01:00:00.000,sym=`IBM]
1949 787024                                          /roughly 2 secs
```

- Always put date first in the where clause
- NEVER run a query on a historical database without a date in the where clause, the example below shows the massive difference in time:

```
q)\ts do[5;select from trades]       /without where
2970 318769456
q)\ts do[5;select from trades where date=2013.06.09]
20 16778928                            /with where
```


email:info@aquaq.co.uk

Attributes and Queries

- Doesn't have significant improvement on unique column

```
q)quotes:quotes,'([]uni:-1000000?1000000)
q)gquotes:gquotes,'([]`g#uni:-1000000?1000000)
q)pquotes:pquotes,'([]`p#uni:-1000000?1000000)
q)
q)\ts do[1000;select from quotes where uni=5555]
2198 1048992
q)\ts do[1000;select from gquotes where uni=5555]
2051 1048992
q)\ts do[1000;select from pquotes where uni=5555]
2174 1048992
```

- Attributes are lost when data is extracted from the table:

```
q)meta select from gquotes where sym=`IBM
c    | t f a
-----| -----
time | t
sym  | s
bid  | f
ask  | f
bsize| j
asize| j
uni  | j
```


email:info@aquaq.co.uk

Exercises: Tables and Queries

Please refer to kdbexercises.pdf

Complete sections 7.1 and 7.2 on Tables and Queries.

email:info@aquaq.co.uk

Joins

email:info@aquaq.co.uk

Joins

- Joins combine data together from different tables.
- Some joins are **keyed**, i.e. the join is matched on columns in a keyed table.
- Some joins are **asof**, i.e. the time column in the first table dictates intervals for the time column in the second table.
- Columns will be filled with nulls if necessary.
- For the following examples, trades and quotes tables have been created using the fakedb.q script.

Vertical Joins

table1 , table2

- The vertical join ‘,’ is used as for joining simple lists, and **table2** is concatenated to **table1**.
- Both tables must have exactly the same schema (column names and column types) or the join will fail.

Vertical Joins

```
q)T1:3#trades
q)T2:-3#trades
q)T1
time         sym  side price size
---------------------------------
00:00:09.377 IBM  buy  43.51 5000
00:00:10.395 NOK  buy  31.78 5000
00:00:10.465 CSCO sell 35.43 1500
q)T2
time         sym  side price size
---------------------------------
23:59:33.426 NOK  buy  30.52 1000
23:59:36.749 NOK  sell 30.49 1000
23:59:39.839 YHOO buy  33.48 500
q)T1,T2
time         sym  side price size
---------------------------------
00:00:09.377 IBM  buy  43.51 5000
00:00:10.395 NOK  buy  31.78 5000
00:00:10.465 CSCO sell 35.43 1500
23:59:33.426 NOK  buy  30.52 1000
23:59:36.749 NOK  sell 30.49 1000
23:59:39.839 YHOO buy  33.48 500
```


email:info@aquaq.co.uk

Horizontal Join

table1 ,' table2

- The horizontal join joins tables with the same number of rows; i.e. adds an extra set of columns.

Horizontal Join

```
q)show table1: select time, sym from trades
time         sym
-----------------
00:00:25.251 DELL
00:00:32.813 ORCL
00:00:47.925 ORCL
00:00:51.357 IBM
...
q)show table2: select price,size from trades
price size
----------
41.33 3500
36.38 500
36.4  3500
30.42 500
...
q)
q)table1,'table2
time         sym  price size
----------------------------
00:00:25.251 DELL 41.33 3500
00:00:32.813 ORCL 36.38 500
00:00:47.925 ORCL 36.4  3500
00:00:51.357 IBM  30.42 500
...
```

Left Join

table lj **keyedtable**

- A left join is a horizontal join, based on the value of a subset of columns.
- The right argument must be keyed, and where the keys are matched to the columns in the left argument the join is completed. The left argument can be keyed or unkeyed.
- If a key from **keyedtable** isn't present in **table**, that data will not be joined.
- If there is no key present in **keyedtable** which is in **table**, then the join results in null values.

email:info@aquaq.co.uk

Left Join

- A left join is most commonly used to join reference or static data to time series data; for example:

```
q) trades
time         sym  side price size
---------------------------------
00:00:09.377 IBM  buy  43.51 5000
00:00:10.395 NOK  buy  31.78 5000
00:00:10.465 CSCO sell 35.43 1500
...
q) select countTrades: count i by sym from trades
sym | countTrades
----| -----------
AAPL| 1462
CSCO| 1430
DELL| 1371
...
q) trades lj select countTrades: count i by sym from trades
time         sym  side price size countTrades
---------------------------------------------
00:00:09.377 IBM  buy  43.51 5000 1456
00:00:10.395 NOK  buy  31.78 5000 1462
00:00:10.465 CSCO sell 35.43 1500 1430
.../- Here, the tables are joined on `sym as the key
```

Plus Join

table pj **keyedtable**

- A plus join (pj) is a variation on the left join;
- Where the key column names match, numeric values are added.
- Other columns of the left argument remain unchanged.
- For more information on this join, see: `http://code.kx.com/wiki/Reference/joins`.

email:info@aquaq.co.uk

Plus Join

```
q)show T1:select volume:sum size,avgprice: avg price by sym  from
 trades1
sym | volume  avgprice
----| ----------------
AAPL| 2170500 24.4455
CSCO| 2038000 34.52388
DELL| 1958500 29.58848
...
q)
q)show T2:select volume:sum size by sym from trades2
sym | volume
----| -------
AAPL| 2130500
CSCO| 2062500
DELL| 2024500
...
q)
q)T1 pj T2
sym | volume  avgprice
----| ----------------
AAPL| 4301000 24.4455
CSCO| 4100500 34.52388
DELL| 3983000 29.58848
...
```


email:info@aquaq.co.uk

Union Join

table1 uj **table2**
keyedtable1 uj **keyedtable2**

- A union join allows tables with different columns to be joined, and is an extension of the simple vertical join (‘,’).
- Both tables must be either keyed or unkeyed.
- When both tables are **unkeyed**:
  - The resulting table is simply the superset of rows and columns of the two tables;
  - The length of the new table is equal to the length of **table1** plus the length of **table2**;
  - If the columns are shared, they are concatenated;
  - If the columns are not shared, they will be filled with nulls.

email:info@aquaq.co.uk

Union Join

```
q)`a xasc ([]a:til 5;b:5?`1;c:5?`1) uj ([]a:2*til 5;b:5?`2;d:5?`2)
a b  c d
---------
0 g  p
0 fp   ij
1 j  k
2 a  i
2 ol   ap
3 n  n
4 a  n
4 pf   ml
6 ll   fm
8 cn   ii
```

- When both tables are **keyed**:
  - Update existing records in **keyedtable1** with matches from **keyedtable2**;
  - Where no match is found, the row is appended.
  - For example, if we want to join two tables which are time bucketed, we can use a uj -

email:info@aquaq.co.uk

Union Join

```
q)show t1:select tradect:count time by 4*floor time.hh%4 from trades
x | tradect
--| -------
8 | 950
12| 925
16| 125
q)show t2: select quotect:count time by 4*floor time.hh%4 from quotes
x | quotect
--| -------
8 | 4740
12| 4679
16| 581
q)t1 uj t2
x | tradect quotect
--| ---------------
8 | 950     4740
12| 925     4679
16| 125     581
```


 email:info@aquaq.co.uk

Inner Join

table ij **keyedtable**

- An inner join joins two tables on the keys of **keyedtable**.
- Where a match occurs, the column is either added on, or updated if it already exists - the result will be a table with one row for each row of the first table which matches the keys of the second.
- Non-matches are not returned in the result - i.e. there will be no nulls in the resulting table.
- Equi join (ej) is a variation on ij, where the column names can be specified. For more information on this join, see: `http://code.kx.com/wiki/Reference/joins`.

email:info@aquaq.co.uk

Inner Join

```
q) show t1:select volume:sum size by sym from trades
sym | volume
----| -------
AAPL| 4051500
CSCO| 3924000
DELL| 4149500
IBM | 4098000
q)show t2:select TradeVal:1e-6*sum size*price by sym from trades
sym | TradeVal
----| --------
AAPL| 14.2685
CSCO| 14.96273
DELL| 13.37905
GOOG| 29.6953
IBM | 16.72193
q)t1 ij t2
sym | volume TradeVal
----| ---------------
AAPL| 498899 14.2685
CSCO| 591248 14.96273
DELL| 602919 13.37905
GOOG| 652281 29.6953
IBM | 560199 16.72193
```

Asof Join

aj[`col1`col2; **table1**; **table2**]
aj0[`col1`col2; **table1**; **table2**]

- An asof join is not a straight match on a column value. Instead, the values in the chosen column in **table1** give an interval within which to match from **table2**.
- The last value within the interval is taken from **table2**.
- If the resulting time value is to be taken from **table2** rather than **table1**, then use **aj0**.
- A common usage is for time series data, when the time values will not match up exactly - instead, an asof join will select the row from **table2** with the time not more than the corresponding time in **table1**. It joins the last value from **table2** to **table1**.

email:info@aquaq.co.uk

Asof Join

- The first argument `col1`col2 is a list of columns to join on, commonly `sym`time.
- The last column in the list is the **asof** match. The remaining column(s) are the exact match columns.
- For correct and optimum performance, it is recommended that:
  - there should be at most one exact match column
  - the exact match column should have a `p or `g attribute set
  - the data **must** be sorted by the asof column within the match column (so in the `sym`time example, the data doesn't have to be globally sorted by time, but does have to be sorted within each sym)

Asof Join

```
q)trades
time         sym  side price size
---------------------------------
00:00:09.377 IBM  buy  43.51 5000
00:00:10.395 NOK  buy  31.78 5000
00:00:10.465 CSCO sell 35.43 1500
00:00:11.424 CSCO sell 35.43 1000
...
q)quotes
time         sym  bid   ask   bsize asize
-----------------------------------------
00:00:00.152 ORCL 32.17 32.22 10000 7000
00:00:00.577 CSCO 35.45 35.5  1500  1500
00:00:02.157 DELL 29.03 29.05 8000  2000
00:00:02.834 AAPL 25.32 25.35 3500  7500
...
q)aj[`sym`time;trades;quotes]
time         sym  side price size bid   ask   bsize asize
---------------------------------------------------------
00:00:09.377 IBM  buy  43.51 5000 43.49 43.51 8000  9000
00:00:10.395 NOK  buy  31.78 5000 31.77 31.78 6000  9000
00:00:10.465 CSCO sell 35.43 1500 35.43 35.47 3000  7000
00:00:11.424 CSCO sell 35.43 1000 35.43 35.47 3000  7000
...
```

Window Join

wj[w; c; **table1**; (**table2**; (f0;c0); (f1;c1))]

- A window join is a generalisation of the asof join.
- A window join aggregates values within an interval; where an asof join would join the most recent value from quotes to trades, a window join would join an aggregation of the available quotes a given time interval; e.g. in the fifteen minutes before the trade occurred.
- The arguments are as follows:
  - **table1** and **table2** are the unkeyed tables to join;
  - **w** defines the time interval - a pair of times or timestamps;
  - **c** the common columns to join on (`sym`time);
  - **f0** and **f1** are the aggregation functions to apply to columns **c0, c1** over **w**.

Window Join

```
q) /-create arguments to wj -
q) window:-5000 5000+\:trades.time
q) f:`sym`time
q)
q) /- Apply avg to bid and ask in quotes
q)wj[window;f;trades;(quotes; (avg;`bid);(avg;`ask))]
time         sym  side price size bid      ask
---------------------------------------------------
00:00:25.251 DELL sell 41.33 3500 36.74259 36.77519
00:00:32.813 ORCL sell 36.38 500  37.19529 37.23235
00:00:47.925 ORCL buy  36.4  3500 36.53077 36.56308
00:00:51.357 IBM  sell 30.42 500  33.98583 34.01833
00:00:56.316 IBM  sell 30.42 500  30.42    30.43
...
q)/- We can see the values used for the aggregation:
q)wj[window;f;trades;(quotes; (::;`bid);(::;`ask))]
time         sym  side price size bid
--------------------------------------------------------------------
00:00:25.251 DELL sell 41.33 3500 41.33 35.93 30.44 27.19 46.86  ...
00:00:32.813 ORCL sell 36.38 500  36.38 41.34 27.18 41.31 36.38 ...
00:00:47.925 ORCL buy  36.4  3500 36.37 32.18 46.92 41.36 46.9 ...
00:00:51.357 IBM  sell 30.42 500  30.43 36.38 27.17 27.16 46.9 ...
00:00:56.316 IBM  sell 30.42 500  ,30.42
...
```


email:info@aquaq.co.uk

Exercises: Joins

Please refer to kdbexercises.pdf

Complete chapter 8 on Joins.

email:info@aquaq.co.uk

Adverbs

email:info@aquaq.co.uk

Adverbs

- Combine 2 verbs and use it as a new function
- Eachboth (')

```
q)(1 2 3),'(1 2 3)     /join each right and left
1 1
2 2
3 3
```

- If a function can accept lists, its better to not use each:

```
q)f:{x+y}
q)f'[1 2;3 4]
4 6

q)f[1 2;3 4]
4 6
```

- Eachleft (\:)

```
q)1 2 3,\:1 2 3    /add right to each left
1 1 2 3
2 1 2 3
3 1 2 3
```


email:info@aquaq.co.uk

Adverbs

- Eachright (/:)

```
q)1 2 3,/:1 2 3    /minus left to each right
1 2 3 1
1 2 3 2
1 2 3 3
```

- Eachleft+eachright (\:)

```
q)1 2 3,\:/:1 2 3    /add right to each left
1 1  2 1  3 1
1 2  2 2  3 2
1 3  2 3  3 3
q)raze 1 2 3,\:/:1 2 3    /add right to each left
```

- over (/)

```
q)*/[1 2 3]                 /f[x n;f[x 2;f[x 1;x 0]]..]
6
q){x+y+z}/[1 5 6;2 22;3 33]   /f[f[..f[f[x;y 0;z 0];y 1;z 1]..];y n;z n]
61 65 66
q){x,sum -2#x}/[10;0 1]       /f[..f[f[x]]..] n times
0 1 1 2 3 5 8 13 21 34 55 89
```


email:info@aquaq.co.uk

Adverbs

- scan (\)

```
q)*\[1 2 3]                    /x 0,f[x 0;x 1],f[f[x 0;x 1];x 2],...
1 2 6
q){x+y+z}\[1 5 6;2 22;3 33]    /f[x;y 0;z 0],f[f[x;y 0;z 0];y 1;z 1],...
6  10 11
61 65 66
```

- prior (':)

```
q)-':[1 4 2]                   /f[x 0;0N],f[x 1;x 0],f[x 2;x 1],...
1 3 -2
q)deltas 1 4 2                 /same as deltas
1 3 -2
```

For more information:

`http://code.kx.com/wiki/Reference`

Back to project Euler...

- Now that we know a bit more q, we can improve on the earlier solution quite a bit
- Find the sum of all the multiples of 3 or 5 below 1000:

```
q)sum where max 0=mod/:[;3 5]til 1000
233168
```


 email:info@aquaq.co.uk

Exercises: Adverbs

Please refer to kdbexercises.pdf

Complete chapter 9 on Adverbs.

email:info@aquaq.co.uk

Advanced Adverbs

email:info@aquaq.co.uk

Scan and Over

- The two most complex adverbs are scan and over
- The behaviour changes depending on function valence and the types of the parameters
- The examples in this section will mainly be with scan as it is clearer. The result of over is usually the last value from the result of scan
- The usual use case is iterative operations - to propagate forward the result of a previous calculation

email:info@aquaq.co.uk

Scan and Over

```
// Monadic function, integer first argument - iteration
// x value is whole list
q){x-1}\[3;10 11 12]
10 11 12
9  10 11
8  9  10
7  8  9

// Monadic function, functional first argument - while
q){x-1}\[{max[x] > 10};10 11 12]
10 11 12
9  10 11
8  9  10

// Monadic function, single argument - iterate until input=output
q){x%10}/[10]
0f
q){x%10}\[10]
10
1f
0.1
0.01
0.001
0.0001
1e-05
..
```


email:info@aquaq.co.uk

Scan and Over

```
// diadic function, two arguments
// first argument is initial value
// x is result of previous function, y is next value
q){x+y}\[20 21;4 5 6]
24 25
29 30
35 36

// multivalent is similar
q){x+y*z}\[20 21;4 5 6;20 30 40]
100 101
250 251
490 491

// any inbuilts that use adverbs can also utilise these features
q)deltas
-':
q)x:98 99 100 99
q)deltas x
98 1 1 -1
q)deltas[0N;x]
0N 1 1 -1
q)deltas[first x;x]
0 1 1 -1
```

Scan and Over

- We can create a function to add daily interest to a capital value like this:

```
q)addinterest:{[c;r] c*1+r%36500}
// add 3% daily interest to 1200
q)addinterest[1200;3]
1200.099
```

- Assuming that interest is compounded, use adverbs to:
    - Calculate the capital value of 1200 at 3% interest after 4 days
    - Calculate how many days it takes the capital value to increase to 1300
    - The interest rate changes to 3 days at 3%, 2 days at 4%, then 3 days at 5%. Calculate the capital value at the end of the 8 days.
    - Modify the previous calculation, but calculate for initial capital values of 1200, 1500 and 2000

Scan and Over

```
q)addinterest[;3]\[4;1200]
1200
1200.099
1200.197
1200.296
1200.395
q)addinterest[;3]/[4;1200]
1200.395
q)count addinterest[;3]\[1300>;1200]
975

q)addinterest/[1200;3 3 3 4 4 5 5 5]
1201.052
q)addinterest\[1200;3 3 3 4 4 5 5 5]
1200.099 1200.197 1200.296 1200.427 1200.559 1200.723 1200.888 1201.052

q)addinterest/[1200 1500 2000;3 3 3 4 4 5 5 5]
1201.052 1501.316 2001.754
q)addinterest\[1200 1500 2000;3 3 3 4 4 5 5 5]
1200.099 1500.123 2000.164
1200.197 1500.247 2000.329
1200.296 1500.37  2000.493
1200.427 1500.534 2000.712
1200.559 1500.699 2000.932
1200.723 1500.904 2001.206
1200.888 1501.11  2001.48
1201.052 1501.316 2001.754
```


email:info@aquaq.co.uk

Book Building

- Sometimes it is necessary to work on 'book' data
- Book data is multiple different orders or quotes at different price points from different market participants or trading venues
- It is relatively straight forward to calculate the book at any given point in time
- It is sometimes necessary to calculate the book at every time point, which can be very time consuming
- To calculate the book you need to take account of the current order/quote and modify/replace it in the current book
- In our example, we will work with quotes from different venues, so we can key the book dictionary by venue

email:info@aquaq.co.uk

Book Building

```
// create a function to add a price, return book
q)add:{[book;ex;p] book[ex]:p; book}

// use over to get the final book
q)add/[()!();`a`b`a`a;5 6 3 2]
a| 2
b| 6
// or scan to get all the books
q)add\[()!();`a`b`a`a;5 6 3 2]
(,`a)!,5
`a`b!5 6
`a`b!3 6
`a`b!2 6

// create a buildbook function
q)bb:{add\[()!();x;y]}
// add the book to a table
q)update bidbook:bb[src;bid] by sym from
select sym,src,bid,bsize from quotes
sym  src bid   bsize bidbook
------------------------------------------
MSFT L   36.02 3000  (,`L)!,36.02
YHOO O   35.51 3500  (,`O)!,35.51
MSFT N   36.03 8500  `L`N!36.02 36.03
MSFT O   36    6500  `L`N`O!36.02 36.03 36
ORCL L   32.2  3000  (,`L)!,32.2
..
```


email:info@aquaq.co.uk

Book Building

- You can then analyze this data, e.g. calculate the best bid at each time point

```
q)\ts t1:update bestbid:max each bidbook from
update bidbook:bb[src;bid] by sym from
select sym,src,bid,bsize from quotes
97 15107024
q)t1
sym  src bid   bsize bidbook               bestbid
--------------------------------------------------
MSFT L   36.02 3000  (,`L)!,36.02          36.02
YHOO O   35.51 3500  (,`O)!,35.51          35.51
MSFT N   36.03 8500  `L`N!36.02 36.03      36.03
MSFT O   36    6500  `L`N`O!36.02 36.03 36 36.03
ORCL L   32.2  3000  (,`L)!,32.2           32.2
..
```

- However, building all the books may be unnecessary
- If you only want the best bid, then you can calculate that directly

 email:info@aquaq.co.uk

Book Building

```
// calculate the bid book, and find the max
// return a list of (currentbook; best bid)
q)addbid:{[c;ex;p] c[0],:(enlist ex)!enlist p; (c[0];c[1],max c[0])}
q)bbid:{last addbid/[(()!();());x;y]}

q)update bestbid:bbid[src;bid] by sym from
select sym,src,bid,bsize from quotes
sym  src bid   bsize bestbid
----------------------------
MSFT L   36.02 3000  36.02
YHOO O   35.51 3500  35.51
MSFT N   36.03 8500  36.03
MSFT O   36    6500  36.03
ORCL L   32.2  3000  32.2
..

// In this case, it is slower but less memory
q)\ts update bestbid:bbid[src;bid] by sym from
select sym,src,bid,bsize from quotes
301 5506528
```


email:info@aquaq.co.uk

Protected Execution

email:info@aquaq.co.uk

Protected Execution

- Similar to try-catch in C++
- Avoids crashing during execution.
- An exception can be handled or redirected
- For monadic function, use @

```
q)f:{x+1}
q)@[f;1;`$"not valid input"]
2
q)@[f;1 2 3;`$"not valid input"]
2 3 4
q)@[f;`a;`$"not valid input"]
`not valid input
```

- For multivalent function, use .

```
q)g:{x+y*z}
q).[g;1 2 3;`$"not valid input"]
7
q).[g;(1;2;3;4);`$"not valid input"]
`not valid input
q).[g;(1;2;`a);`$"not valid input"]
`not valid input
```


email:info@aquaq.co.uk

Protected Execution

- Handler is evaluated even if no errors occur. Wrap in a function to avoid this:

```
q)@[f;1;'error]
'error
q)@[f;1;{'error}]
2
```

The input to the error function is the error text:

```
q)@[f;`1;{'"error: ",x}]
'error: type
```

You can get creative with error handling:

```
q)if[0b=res:@[f;1;0b];0N!"error evaluating expression"]

q)if[0b=res:@[f;`a;0b];0N!"error evaluating expression"]

"error evaluating expression"
```


email:info@aquaq.co.uk

Protected Execution

For more information:

`http://code.kx.com/wiki/JB:QforMortals/execution_control`

email:info@aquaq.co.uk

Namespaces

email:info@aquaq.co.uk

Namespaces

- The system command:

```
\d
```

is used to set the **current namespace** (also known as the directory or context)

- It is possible to switch the current namespace to those defined by users or by Kx:

```
\d .Q
\f
`Cf`IN`L`MAP`S`V`addmonths`addr`bv`chk`cn`d0`dd`def`dpft..
```

- One can return to the default namespace by using:

```
\d .
```


 email:info@aquaq.co.uk

Namespaces

- Moving to a new, undefined namespace can be done using the same syntax:

```
\d .newns
```

  However the namespace will not have been created until an object is defined within it:

```
\d .newns
a:1
```

- This can be checked using the command:

```
key`
`q`Q`h`o`newns
```

- Namespaces are implemented internally as dictionaries:

```
q).newns
```


email:info@aquaq.co.uk

Namespaces

- Global variable scoping in dictionaries are restricted to the namespace you are currently in

```
q)\d .newns
q.newns)f:{a+1}
q.newns)a:10
q.newns)\d .
q).newns.f[]
11
q)a:0
q).newns.f[]
11
q).newns.g:{a+1}
q).newns.g[]
1
```

System Commands

email:info@aquaq.co.uk

Command line operators

When runing Q from the command line, it can take many parameters:

q [f] [-b] [-c r c] [-C r c] [-e 0|1] [-g 0—1] [-l] [-L][-o N] [-p N] [-P N] [-q] [-r :H:P] [-s N] [-t N] [-T N] [-u|U F] [-w N] [-W N] [-z 0|1]

- **-b** : Blocks client write access to a database.

```
os> q -b
q)\_      /check to see if write access is blocked
1b
```

- **-e 0|1** : Enable client error trapping.
- **-g 0|1** : Calls garbage collection. 1 triggers immediate collection, and 0 defers collection.
- **-o N** : The local time offset; N hours from GMT.
- **-s N** : Use N number of slaves.
- **-t N** : Timer in N milliseconds between timer ticks. Default is 0, for no timer.

email:info@aquaq.co.uk

Command line operators

- **-T N** : Set timeout for client queries where N is in milliseconds between timer clicks. default is 0.
- **-u 1** : Disable system exits.
- **-w N** : Workspace size limit, default is 2*RAM.
- **-W N** : Start of week as an offset from saturday. Default is 2, meaning that Monday is the start of week.
- **-z 0|1** : Date format parsing, 0 is mm/dd/yyyy, 1 is dd/mm/yyyy.

email:info@aquaq.co.uk

System commands

KDB can be given system commands directly from the q prompt. System commands are called using either \x or 'system "x"'.

- **\b** : Lists all dependencies/views.
- **\B** : Lists all pending views. These are dependencies that have not yet been referenced.

```
q)a::x+1                    / a depends on x
q)\B                        / the dependency that is pending
,`a
q)x:10
q)\B                        / still pending after x is defined
,`a
q)a                         / use a
11
q)\B                        / no longer pending
`symbol$()
```

- **\c [h,w]** : Console height and width.
- **\P N** : Float precision; sets the number of float numbers displayed.

email:info@aquaq.co.uk

System commands

- **\S** : Sets the random number seed.

```
q)5?10
8 1 9 5 4
q)5?10
6 6 1 8 5
q)\S -314159      / restore default seed
q)5?10            / same random numbers generated
8 1 9 5 4
```

- **\t N** : Sets the timer; where N is an integer parameter of the number of milliseconds between timer ticks. If 0, the timer is turned off.
- **\ts exp** : Time and space of expression. Executes the expression and shows the execution time in milliseconds and the space used in bytes.
- **\x .z.p*** : Expunge handler, restores the .z function to it's orginal value.
- **\1 *.txt** : Redirect stdout to *.txt.

email:info@aquaq.co.uk

System commands

- **\2 *.txt** : Redirect stderr to *.txt.

email:info@aquaq.co.uk

.z.*

The .z namespaces contains a number of useful functions for common tasks.

- **.z.exit** : Provides an action on exit; .z.exit is called with the exit parameter as the argument just before the session exits.

```
q).z.exit:{0N!x}
q)exit 42
42
os>..
q)\x .z.exit  /unset
```

- **.z.ts** : The function that is invoked at intervals set by \t.

```
\t 1000
.z.ts:{0N!x} /x is current time
17:12:12.849442000
17:12:13.849442000
```


email:info@aquaq.co.uk

.z.*

- **.z.vs** : Once .z.vs is defined, it is invoked with two arguments after a variable has been set. The first argument is the symbol of the variable that is being modified, and the second is the index.

```
q).z.vs:{0N!(x;y;value x)} //setting vs
q)m:(1 2;3 4) /setting a value
(`m;();(1 2;3 4)) /vs is called and returns variable and index
q)m[1;1]:0
(`m;1 1;(1 2;3 0)) /vs is called and returns variable and index
```

- **.z.W** : Returns a dictionary of ipc handles with the number of bytes waiting in their output queues.
- **.z.P** : System localtime timestamp to nanoseconds
- **.z.p** : UTC timestamp to nanoseconds
- **.z.Z** : System localtime as datetime
- **.z.z** : UTC as datetime
- **.z.[tTdD]** : Time/date shortcuts.

.z.*

- **.z.zd** : Can be defined as an integer list of default parameters for logical block size, compression algorithm and compression level that apply when saving to binary files.

```
q).z.zd:17 2 6          / set zip defaults
q)\x .z.zd              / unset
```

- **.z.ac** : Http authenticate from cookie. Allows users to define custom code to extract Single Sign On (SSO) token cookies from the http header and verify it, decoding and returning the username, or instructing what action to take.

```
q).z.ac:{mySSOAuthenticator x[1]`Authentication}
```

- **.z.bm** : IPC Message Validator. When recieving a badly formed message .z.bm is called with a single arg;. a list of (handle;msgBytes)

Attributes

email:info@aquaq.co.uk

Attributes

- kdb+ has 4 attributes which can be applied to datastructures (usually lists):
  - **sorted** (`s): items are in ascending order
  - **parted** (`p): items are contiguous (though not necessarily in order)
  - **unique** (`u): each element is unique
  - **grouped** (`g): a mapping from distinct item to each index is maintained
- Each attribute provides different performance benefits
- Attributes may be lost upon modification

email:info@aquaq.co.uk

Sorted

- Donates a list as sorted ascending
- Improves search performance (binary search rather than linear) and min/max change to first/last
- Is maintained on sorted append, lost on other modification

```
q)a:100000?10000000
q)b:asc a
q)b
`s#27 64 136 152 368 418 565 577 674 881 883 89..
q)q)\t:1000 max a
98
q)q)\t:1000 max b
0
q)b,max b
27 64 136 152 368 418 565 577 674 881 883 895 9..
q)b,:max b
q)b
`s#27 64 136 152 368 418 565 577 674 881 883 89..
q)b,:min b
q)b
27 64 136 152 368 418 565 577 674 881 883 895 9..
q)`s#a
's-fail
```

Parted

- List is parted with elements stored contiguously
- Breakpoints of elements stored - no more searching, contiguous reads
- Lost on any modification. Mainly used for on-disk queries
- u-fail error if list not sorted

```
q)a:1000000?`$'.Q.a
q)b:`p#asc a
q)\t:1000 where a=`m
1038
q)\t:1000 where b=`m
777
q)b
`p#`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a..
q)b,:`z`z
q)b
`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`a`..
q)`p#a
k){$[3=x;(`#y;`u#y i;(i:&~=':y),#y);(y;`u#!r;+\0,#:'x;,/x:. r:=y)]}
'u-fail
#
`u
`a`i`s`u`c`s`i`y`k`w`u`w`i`y`z`y`g`k`z`j`l`x`i`..
```


email:info@aquaq.co.uk

Unique

- Signifies each element of the list is unique
- Hash function replaces searches - constant time lookups
- Mainly used for single keyed tables and big dictionaries
- Maintained on unique append

Unique

- u-fail error if list not unique

```
q)a:-100000?100000
q)b:`u#a
q)a[0 50000 99999]
97165 5588 48683
q)\t:10000 a?97165
1
q)\t:10000 a?5588
201
q)\t:10000 a?48683
401
q)\t:10000 b?97165
2
q)\t:10000 b?5588
2
q)\t:10000 b?48683
3
q)b
`u#97165 11238 96129 7617 24630 50128 52193 522..
q)b,:-100000
q)b
`u#97165 11238 96129 7617 24630 50128 52193 522..
q)`u#b,1
'u-fail
```

Grouped

- A separate mapping of unique!indices is maintained
- No searching, no requirement on order or content
- Uses additional memory
- Maintained on append

```
q)a:1000000?`$'.Q.a
q)b:`g#a
q)\t:100 group a
627
q)\t:100 group b
0
q)group b
g| 0 36 41 80 130 164 166 176 202 250 252 258 2..
m| 1 32 46 88 101 180 232 300 312 323 326 340 3..
w| 2 5 19 20 27 35 39 48 61 103 119 163 168 198..
n| 3 12 45 47 57 79 99 116 118 156 229 391 404 ..
p| 4 8 10 53 64 67 75 76 102 112 115 144 146 16..
..
q)b,:`A`B
q)b
`g#`g`m`w`n`p`w`t`z`p`e`p`v`n`q`s`h`k`q`h`w`w`f..
```


email:info@aquaq.co.uk

Special Case: Step Function

- sorted attribute can be used to create a step function from a table or dictionary
- Lookups become ‘asof’, taking the prevailing value

```
q)d:1 1.6 1.9 2.1!`a`b`c`d
q)d[1.0]
`a
q)d[1.1]
`
q)d:`s#d
q)d[1.1]
`a
q)t:`s#([sym:`A`A`B;date:2014.01.01 2014.02.01 2014.05.01]
          val:1.8 1.9 6.1)
q)t(`A;2014.01.06)
val| 1.8
q)t(`B;2014.01.06)
val|
q)t(`B;2015.01.06)
val| 6.1
```


email:info@aquaq.co.uk

IPC

email:info@aquaq.co.uk

IPC

- Tcp/ip based
- 2 ways to start a session listening on a port:
  1. In a q session

     ```
     q)\p 5010
     ```

  2. From the command line

     ```
     >q -p 5010
     ```

- To connect to the server, use the 'hopen' command

  ```
  q)h:hopen 5010
  q)\netstat -an | find /i "5010"                /using system command
  "  TCP    0.0.0.0:5010           0.0.0.0:0              LISTENING"
  "  TCP    127.0.0.1:5010         127.0.0.1:54763        ESTABLISHED"
  "  TCP    127.0.0.1:54763        127.0.0.1:5010         ESTABLISHED"
  ```

  with 'h' the name of the handle.
- To close the connection, use the 'hclose' command

  ```
  q)hclose h
  ```


email:info@aquaq.co.uk

IPC

- Note 1: there is a special loopback handle 0

```
q)0"1+1"
2
q)0"a:1"
q)a
1
```

- Note 2: all connections are bi-directional. If A connects to B, kdb automatically exposes a connection from B to A. The handle for this is .z.w

```
q)show h:hopen 5010 / open a connection to a server
344i                / this process uses handle #344 to communicate
q)h".z.w"           / check out the bi-directional handle
404i                / the remote process uses #404 to communicate back to us
```


email:info@aquaq.co.uk

IPC

- Different parameters can be supplied to hopen:

```
hopen port | `:host:port | `:host:port:user | `:host:port:user:password
hopen `:unix://port | `:unix://port:user | `:unix://port:user:password
/domain sockets (since 3.4)
hopen (`::port:user:password;timeoutmillisec)
```

- There are three message types:

  1. Synchronous

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

  2. Asynchronous

     ```
     q)neg[h]"a:2"
     ```

  3. Response: this is sent automatically by the remote process on completing a sync request.

- There are two message formats:

  1. String format

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

IPC

2. Functional format

```
q)h(+;2;2) / similar to postfix notation +[2;2]
```


email:info@aquaq.co.uk

IPC

- Some basic message handlers:
  - **.z.po** - called when a connection to a kdb+ session has been initialized
  - **.z.pc** - called after a connection has been closed
  - **.z.pg** - called when the session receives a synchronous message
  - **.z.ps** - called when the session receives an asynchronous message
- Message handlers can be reset to default using \x
- Asynchronous messages can be queued
- Asynchronous flush:

```
q)neg[h][]        /To block until all async messages are sent
q)neg[h](::)
```


email:info@aquaq.co.uk

IPC

- To confirm async messages are received and processed, chase with sync message

```
q)h""
```


email:info@aquaq.co.uk

IPC

- Note that sending an async message does not send the message immediately:

```
neg[h]x
```

- It serializes x, and queues it for sending at a later time - either when the main loop spins, or a blocking request is issued on that handle, such as:

```
h""
```

- This is a sync message which first flushes any pending outgoing messages on h, sends the request message and then processes any pending incoming messages on h (and blocks) until a response message is received.
- To block until all pending outgoing messages have been written into the socket:

```
neg[h][]
```

IPC

- To flush any pending outgoing messages on h, and block until any message is received on h:

```
h[]
```

- Hence, if you want to ensure an async message really has been sent as soon as is possible, use:

```
neg[h]x; neg[h][];
```


email:info@aquaq.co.uk

Example

- Executing functions from one session to another

```
q)/starting a new server
q).z.po:{`h set .z.w}
q).z.pc:{show string[.z.h]," dc!"}
q)\p 5010
q)f:{x+2}
q)a
1
q)b
2
q)
q)
q)
q)
q)
q)
q)h"f[11]"        /string form
22
q)h(`f;11)        /remote function
22
q)h(f;11)         /local function
13
q)"kent-pc dc!"
```

```
q)/connecting to the server
q)
q)
q)h:hopen 5010
q)f:{x*2}
q)h"a:1"          /synchronous
q)
q)neg[h]"b:2"     /asynchronous
q)
q)h"f[11]"        /string form
13
q)h(`f;11)        /remote function
13
q)h(f;11)         /local function
22
q)
q)
q)
q)
q)
q)
q)hclose h
```


 email:info@aquaq.co.uk

Exercises:IPC

Please refer to kdbexercises.pdf

Complete chapter 10 on IPC.

email:info@aquaq.co.uk

On-Disk Tables

email:info@aquaq.co.uk

KDB+ Database

- Historical and real-time relational databases
- Handles in-memory data and on-disk data in a similar way
- Choose the table layout for your application
- Most historical databases are partitioned by date
- Each **partition** contains an amount of splayed tables
- Each **splayed table** contains several **flat column files**
- A database is similar to a directory
- Two ways to load a database:
  1. In command prompt

     ```
     >q <path of database>
     ```

  2. In q session

     ```
     q)\l <path to database>
     ```


email:info@aquaq.co.uk

Flat Table

- Saving a **flat table**:

```
`:<root>/<table name> set <table>
```

- For the file name to be the same as the table name:

```
save`:<root>/<table>
```


email:info@aquaq.co.uk

Flat Table

- The table is saved whole on disk

```
q)`:./testtab/flattab set ([]c1:"AB";c2:6 7)
`:./testtab/flattab
q)get `:./testtab/flattab
c1 c2
-----
A  6
B  7
q)flattab2:([]aa:1 2;bb:`a`b)
q)save`:./testtab/flattab2
`:./testtab/flattab2
q)load `:./testtab/flattab2
`flattab2
aa bb
-----
1  a
2  b
```


email:info@aquaq.co.uk

Limitations of a Flat File

- Only suitable for small tables (less than 10,000 rows)
- High memory usage when saving huge datasets as flat from memory
- Time consuming when loading flat table from disk to memory
- Takes up a lot of disk space, especially for large tables
- Cannot extract single column from disk, need to load in the whole table
- Cannot setup slave threads to run queries in parallel
- Flat tables are never mapped into memory, the entire table is loaded into memory

email:info@aquaq.co.uk

Splayed Table

- Splaying a table creates a folder with the file structure:

```
`:<root>/<table name>/ set <table>
```

- Consists of a .d file and the column files

```
q)`:./testtab/splaytab/ set ([]a:1 2 3;b:4 5 6f;c:7 8 9i)
`:./testtab/splaytab/
q)key `:testtab/splaytab/        /key shows the .d file and column names
`.d`a`b`c
```

- .d file holds the order of columns

```
q)get `:testtab/splaytab/.d
`a`b`c
q)`:testtab/splaytab/.d set `c`b`a      /amend column order
`:testtab/splaytab/.d
q)get `:testtab/splaytab/               /column order changed
c b a
-----
7 4 1
8 5 2
9 6 3
```


email:info@aquaq.co.uk

Splayed Table

- Each column file holds the values:

```
q)get `:testtab/splaytab/a
1 2 3
q)get `:testtab/splaytab/b
4 5 6f
q)get `:testtab/splaytab/c
7 8 9i
```

- Add a new column by adding a new file:

```
q)`:testtab/splaytab/d set 101b
`:testtab/splaytab/d
q)`:testtab/splaytab/.d set `a`b`c`d
`:testtab/splaytab/.d
q)get `:testtab/splaytab/
a b c d
-------
1 4 7 1
2 5 8 0
3 6 9 1
```

Splayed Table

- Delete a file using hdel:

```
q)hdel `:testtab/splaytab/d
q)`:testtab/splaytab/.d set `a`b`c
```

- Appending records:

email:info@aquaq.co.uk

Splayed Table

```
q)`:testtab/splaytab/ upsert (1;2f;3i)
`:testtab/splaytab/
q)get `:testtab/splaytab/
a b c
-----
1 4 7
2 5 8
3 6 9
1 2 3
q).[`:testtab/splaytab/;();,;([]a:4 5;b:6 7f;c:8 9i)]
`:testtab/splaytab/
q)get `:testtab/splaytab/
a b c
-----
1 4 7
2 5 8
3 6 9
1 2 3
4 6 8
5 7 9
```


email:info@aquaq.co.uk

Splayed Table

- Sorting table

```
q)`a xasc `:testtab/splaytab/
`:testtab/splaytab/
q)get `:testtab/splaytab          /elements in column "a" now ordered
a b c
-----
1 4 7
1 2 3
2 5 8
3 6 9
4 6 8
5 7 9
```

- Applying attributes

```
q)@[`:testtab/splaytab/;`a;`s#]
`:testtab/splaytab/
q)meta `:testtab/splaytab
c| t f a
-| -----
a| j   s
b| f
c| i
```


 email:info@aquaq.co.uk

Splayed Table Layout

- A splayed table directory is contained beneath the root of the kdb+ database, where the splayed table directory name is the same as the mapped splayed table name
- The file name is the column name. The symbol list containing column names is in the hidden file .d. This .d file allows q to maintain column order.
- The splayed table can be seen as being cut vertically along the columns, these are read in parallel

```
/root
  /tablename     \splayed table directory
    .d           \file with column names
    columnname1   \first column of data
    columnname2    \second column of data
```

- For a table to be splayed, every column must be either a simple list or nested list
- The column elements all must be of the same type (same data size). A table with symbol columns can be splayed only if the symbols are enumerated.

email:info@aquaq.co.uk

Enumeration

- Symbol columns have to be enumerated before splaying the table

```
`:<root>/<table name>/ set .Q.en[`:<root>/;<table>]
```

- This is to avoid scanning different lengths of symbol list items as this will be slow for large columns
- Create a sym file that maps the symbol to the column

```
q)t:([]a:1 2 3;b:`a`b`c;c:4 5 6)
q)`:testtab/splaytab2/ set .Q.en[`:testtab/;t]
`:testtab/splaytab2/
q)key `:testtab/
`flattab`flattab2`splaytab`splaytab2`sym
q)get `:testtab/sym
`a`b`c
q)get `:testtab/splaytab2/b
`sym$`a`b`c                         /enumeration
```

- If sym file is lost, the table is meaningless

Enumeration

- Column b will end up as a list of indices that map to the sym file
- Example 1 of sym file corruption:

```
q)t:([]a:1 5 9;b:`a`b`c;c:010b)
q)t
a b c
-----
1 a 0
5 b 1
9 c 0
q)`:testtab/splaytab3/ set .Q.en[`:testtab/;t]
`:testtab/splaytab3/
q)hdel `:testtab/sym                          /assuming sym file is deleted
`:testtab/sym
q)delete from `.                              /clear all the variables
`.
q)\l testtab                                  /reload the splay table
q)splaytab3
a b c                                         /column b contains only index
-----
1 0 0
5 1 1
9 2 0
```


email:info@aquaq.co.uk

Enumeration

- Example 2 of sym file corruption:

```
q)t:([]a:1 5 9;b:`a`b`c;c:010b)
q)t
a b c
-----
1 a 0
5 b 1
9 c 0
q)`:testtab/splaytab4/ set .Q.en[`:testtab/;t]
`:testtab/splaytab4/
q)symfile:get `:testtab/sym
q)`:testtab/sym set reverse symfile    /rearrange the symbols
`:testtab/sym
q)delete from `.                       /clear all the variables
`.
q)\l testtab                           /reload the splay table
q)splaytab4
a b c                                  /column b contains only index
-----
1 c 0
5 b 1
9 a 0
```


 email:info@aquaq.co.uk

Enumeration

- Example 3 of sym file corruption:

```
q)t:([]a:1 5 9;b:`a`b`c;c:010b)
q)t
a b c
-----
1 a 0
5 b 1
9 c 0
q)`:testtab/splaytab4/ set .Q.en[`:testtab/;t]
`:testtab/splaytab4/
q)\l testtab
q)splaytab4
a b c
-----
1 a 0
5 b 1
9 c 0
q)sym:reverse sym /the in memory sym has been modified
q)splaytab4
a b c
-----
1 c 0
5 b 1
9 a 0
```

Limitations of Splayed Tables

- Every column has to be a simple list or a list of simple lists
- A symbol column has to be enumerated before splaying the table
- Keyed tables cannot be splayed

email:info@aquaq.co.uk

Partitioned Table

- Handling daily data is manageable
- Further partition the splayed table into daily chunks
- Partition table:

```
`:<root>/<date>/<table name>/ set <table>
```

- Partition table with symbols:

```
`:<root>/<date>/<table name>/ set .Q.en[`:<root>/;<table>]
```

- Primitive operations do not work on partitioned tables:

```
q)t[0]
q)select[1] from t
q)`p xasc t
q)0#t
```

- A select query is the only way to access a partitioned table
- The virtual column i doesn't refer to the absolute index, but instead the index of each partition

email:info@aquaq.co.uk

Partitioned Table

- Partitions can be spread across multiple filesystems using par.txt
- par.txt is a text file which sits in the top level directory of a database and lists the filesystems used to store the data
- dbmaint.q on code.kx.com contains utilities to modify on disk tables

```
$ cat hdb/par.txt
/data0/hdb
/data1/hdb
```

- If using par.txt, the hdb top level directory cannot contain further partitions
- kdb+ will use .Q.par to compute the directory to save data to. By default, the data will be round-robin saved across partitions
- Saving across multiple disks can lead to better performance as slaves can access the disks in parallel

email:info@aquaq.co.uk

Partitioned Table

- If adding new file systems, the data may need to rebalanced for optimum performance

email:info@aquaq.co.uk

HDB Related Functions in .Q

- Some functions to work with partition tables:

  .Q.ind - access table using absolute index

```
q)`:./test/2013.01.01/t/ set ([]a:1 2 3;b:`:./test/sym?`a`b`c)
`:./test/2013.01.01/t/
q)`:./test/2013.01.02/t/ set ([]a:1 2 3;b:`:./test/sym?`c`d`e)
`:./test/2013.01.02/t/
q)\l test
q)t
date       a b
--------------
2013.01.01 1 a
2013.01.01 2 b
2013.01.01 3 c
2013.01.02 1 c
2013.01.02 2 d
2013.01.02 3 e
q).Q.ind[t;2 3]
date       a b
--------------
2013.01.01 3 c
2013.01.02 1 c
```

`:./test/sym? creates a sym file that maps the sym column

 email:info@aquaq.co.uk

HDB Related Functions in .Q

.Q.bv - virtually fills missing tables within a partitioned database. Uses the latest partition as a template

email:info@aquaq.co.uk

HDB Related Functions in .Q

```
q)\mkdir 2012.12.31
q)\l .
q)system"mkdir 2012.12.31"
q)\l .
q)t
+`a`b!`t
q)select from t
k){0!(?).@[x;0;p1[;y;z]]}
'./2012.12.31/t/a. OS reports: The system cannot find the path specified.
.
?
(+`a`b!`:./2012.12.31/t;();0b;())
q.Q))\
q).Q.bv[]
q)select from t
date       a b
--------------
2013.01.01 1 a
2013.01.01 2 b
2013.01.01 3 c
2013.01.02 1 a
2013.01.02 2 b
2013.01.02 3 c
```


email:info@aquaq.co.uk

HDB Related Functions in .Q

.Q.chk - fills missing table folders/files within a partitioned database. WAY inferior to .Q.bv on large hdb since it creates a ton of missing files which impacts the entire system

email:info@aquaq.co.uk

HDB Related Functions in .Q

.Q.dpft - save a table splayed to a partition of a database

```
q)t:([]a:5 6 7;b:`x`y`z)
q).Q.dpft[`:.;2013.01.04;`b;`t]
`t
q)\l .
q)t
date       b a
--------------
2013.01.01 a 1
2013.01.01 b 2
2013.01.01 c 3
2013.01.02 c 1
2013.01.02 d 2
2013.01.02 e 3
2013.01.04 x 5
2013.01.04 y 6
2013.01.04 z 7
```


email:info@aquaq.co.uk

HDB Related Functions in .Q

.Q.hdpf - save all tables in the global namespace to historical database and purge all data in the tables

```
q)tab1:([]sym:`X`Y`X`X;price:12 10.5 11 9.5)
q)tab2:([]sym:`X`Y`X`X;size:100 50 150 200)
q).Q.hdpf[0;`:./hdb;2013.01.05;`sym]
q)tab1
sym price
---------
q)\l hdb
q)tab1
date       sym price
--------------------
2013.01.05 X   12
2013.01.05 X   11
2013.01.05 X   9.5
2013.01.05 Y   10.5
q)tab2
date       sym size
-------------------
2013.01.05 X   100
2013.01.05 X   150
2013.01.05 X   200
2013.01.05 Y   50
```


 email:info@aquaq.co.uk

HDB Related Functions in .Q

.Q.par - locates a table on disk

```
q).Q.par[`:.;2013.01.04;`t]
`:/user/kdb/2013.01.04/t
```


email:info@aquaq.co.uk

Common Errors

- Missing table:

```
q)\l problem.q
q)\l db/part1
q)select from trades
k){$[$[0>@b:x 2;1;~`sym~*!b;1;~(in;`sym)~2#c:*x 1];(?). x;#r:,/...
'./2013.01.02/trades/sym: The system cannot find the path specified.
.
?
(+`sym`time`side`price`size!`:./2013.01.02/trades;();0b;())
q.Q))\
q).Q.chk `:.                                 /use .Q.chk to fill
()                                           /missing tables
,`:./2013.01.02
()
q)\l .
q)select from trades
date       sym  time         side price size
--------------------------------------------
2013.01.01 AAPL 00:03:47.552 buy  25.33 3500
2013.01.01 AAPL 00:05:35.900 sell 25.31 500
2013.01.01 AAPL 00:16:54.850 sell 25.28 2000
2013.01.01 AAPL 00:45:14.728 sell 25.31 2000
2013.01.01 AAPL 00:59:00.512 sell 25.32 9000
..
```

Common Errors

- Missing column:

```
q)\l problem.q
q)\l db/part2
q)select from quotes
k){$[$[0>@b:x 2;1;~`sym~*!b;1;~(in;`sym)~2#c:*x 1];(?). x;#r:,/...
'./2013.01.01/quotes/ask: The system cannot find the path specified.
.
?
(+`sym`time`bid`ask`bsize`asize!`:./2013.01.01/quotes;();0b;())
q.Q))\
q)\rmdir /q/s 2013.01.01\quotes\          /better to remove the table
q).Q.chk `:.                              /and backfill properly
()
()
,`:./2013.01.01
q)\l .
q)select from quotes
date       sym  time         bid   ask   bsize asize
----------------------------------------------------
2013.01.02 AAPL 00:01:12.759 40.73 40.76 7500  7500
2013.01.02 AAPL 00:01:57.002 40.76 40.78 8000  7500
2013.01.02 AAPL 00:03:53.466 40.73 40.76 500   500
2013.01.02 AAPL 00:05:55.323 40.75 40.79 5000  7500
..
```

Common Errors

- Missing partition:

```
q)\l problem.q
q)\l db/part3
q)select from trades
date       sym  time         side price size
--------------------------------------------
2013.01.01 AAPL 00:03:43.378 buy  39.8  500
2013.01.01 AAPL 00:10:13.713 sell 39.76 5500
2013.01.01 AAPL 00:13:19.767 buy  39.77 1500
2013.01.01 AAPL 00:25:41.979 sell 39.69 7000
..
q)select from quotes
date       sym  time         bid   ask   bsize asize
----------------------------------------------------
2013.01.01 AAPL 00:01:33.599 39.77 39.8  5500  2500
2013.01.01 AAPL 00:01:41.724 39.77 39.8  4500  1500
2013.01.01 AAPL 00:04:28.307 39.77 39.78 1000  2500
2013.01.01 AAPL 00:05:12.550 39.77 39.81 9500  1500
..
q)select distinct date from quotes
date
----------
2013.01.01                           /need to back fill
2013.01.03                           /missing date
```


 email:info@aquaq.co.uk

Common Errors

- Rogue file:

```
q)\l problem.q
q)\l db/part4
k){if[$[1>@d:!f:-1!x;1;`.d~*d];:.[$[qt d;*|`\:f;`.];();:;d:. f]];d@:&~d
 ."\\cd ",$x;f .q.set'{$[0h>@!x:-1!x;. x;x`]}'f:d@&~(d=`html)|p|s:"."in'
'tt.txt
.:
`:tt.txt
q.Q))\
q)hdel `:tt.txt                            /delete rogue file in root
`:tt.txt
q)\l .
q)select from trades
date       sym  time         side price size
--------------------------------------------
2013.01.01 AAPL 00:17:06.977 buy  44.43 1500
2013.01.01 AAPL 00:24:13.156 buy  44.41 3000
..
q)select from quotes
date       sym  time         bid   ask   bsize asize
----------------------------------------------------
2013.01.01 AAPL 00:00:32.233 44.35 44.4  6000  9500
2013.01.01 AAPL 00:01:06.444 44.36 44.39 3500  4000
..
```

Common Errors

- Missing sym file:

```
q)\l problem.q
q)\l db/part5
q)select from trades
date       sym time         side price size
-------------------------------------------
2013.01.01 1   00:02:33.254 8    24.39 3000
2013.01.01 1   00:28:15.200 8    24.42 500
2013.01.01 1   00:42:45.969 8    24.41 2000
2013.01.01 1   00:50:16.786 7    24.43 10000
2013.01.01 1   00:58:08.499 7    24.44 500
2013.01.01 1   01:09:04.992 8    24.44 2000
..
q)select from quotes
date       sym time         bid   ask   bsize asize
---------------------------------------------------
2013.01.01 1   00:00:41.674 24.35 24.4  4500  4500
2013.01.01 1   00:01:14.937 24.34 24.39 7000  10000
2013.01.01 1   00:05:45.406 24.35 24.38 5000  2000
2013.01.01 1   00:07:20.792 24.35 24.39 7000  6500
2013.01.01 1   00:07:43.261 24.37 24.38 3000  3000
2013.01.01 1   00:09:39.440 24.36 24.4  5000  6500
/whole database needs to be backfilled to restore the sym file
```


email:info@aquaq.co.uk

Exercises: On-disk tables

Please refer to kdbexercises.pdf

Complete chapter 12 on On-Disk Tables.

email:info@aquaq.co.uk

Data Analysis

email:info@aquaq.co.uk

Data Analysis

- kdb+ makes complex operations simple
- The key to making full use of the power and expressiveness of kdb+ is to understand the datastructures
- There only are 3 datastructures: atom, list and dictionary
- A table is a list of dictionaries, and also the 'flip' of a dictionary
- Each column of a table is a list
- Every in-built operator which runs on a list can run on a table column
- The following sections outline some common techniques used in data manipulation

email:info@aquaq.co.uk

Where Clause

- The **where** clause is evaluated left to right, and cascades - conditions are only evaluated on the result of the previous condition
- When working with partitioned data, it is necessary to filter on the partition field first e.g:
  'select from trade where date=X, ... other conditions ..'
- After the partition field, the next field should be any with an attribute e.g:
  'select from trade where date=X, sym=Y, ... other conditions ..'
- If there are multiple fields with attributes to be queried on, the attribute can only really be used in the first where condition

Where Clause

- where conditions should be ordered to cut down the data as fast as possible:

```
q)n:10000000
q)trade:([]price:(n?100f),-1f;size:(n+1)?1000)
q)\t select from trade where size>500,price<0
78
q)\t select from trade where price<0,size>500
13
```

- Due to the cascading nature of the where clause, where conditions are similar to 'and's, but not exactly the same
- Any query where the condition is compared to a 'relative' value will be different depending on order

Where Clause

- In the following example, we want to extract any trades where the size is greater than 500 and the price is equal to the maximum traded price for the full set:

```
q)trade:([]price:50 51 78;size:600 700 300)
q)select from trade where size>500,price=max price
price size
----------
51    700
q)select from trade where price=max price,size>500
price size
----------
```

- The second example is correct - there are 0 trades which match the requirements

Where Clause

- The **in** operator will return true or false depending on whether the left argument is in the right argument
- A common example is with simple lists, e.g:
  'select from trade where sym in SYMLIST, exch in SYMLIST'
  will return the superset of syms and exchs in the two lists
- An alternative requirment might be to only return the rows within a given set of sym and exchange tuples

email:info@aquaq.co.uk

Where Clause

- in can also be used for the following example:

```
q)trade:([]sym:`A`A`B`B;exch:`L`N`L`N;price:1 2 3 4)
q)matches:([]sym:`A`B;exch:`L`N)
q)show trade
sym exch price
--------------
A   L    1
A   N    2
B   L    3
B   N    4
q)matches
sym exch
--------
A   L
B   N

// using 2 in conditions just returns the superset
q)select from trade where sym in matches`sym,exch in matches`exch
sym exch price
--------------
A   L    1
A   N    2
B   L    3
B   N    4
```


email:info@aquaq.co.uk

Where Clause

```
// We can use in on tables, provided the columns match
q)(select sym,exch from trade) in matches
1001b

// and we can drop that into the where condition
q)select from trade where ([]sym;exch) in matches
sym exch price
--------------
A   L    1
B   N    4
```


 email:info@aquaq.co.uk

Where Clause

- **fby** can be used to aggregate data within the where clause
- An example would be to select out all the trades where the price is equivalent to the maximum for the day, for each symbol
- The function used with fby can be inbuilt or user defined, and uniform or aggregative. However, it must be monadic.

 email:info@aquaq.co.uk

Where Clause

```
q)show trade
sym exch price size
-------------------
A   L    20    100
A   N    21    101
B   L    46    102
B   L    45    103
// This is not what we want
q)select from trade where price=max price
sym exch price size
-------------------
B   L    46    102
q)select from trade where price=(max;price) fby sym
sym exch price size
-------------------
A   N    21    101
B   L    46    102
// fby example is equivalent to this
// select from (update mp:max price by sym from trade) where price=mp
// can use fby on multiple columns
q)select from trade where price=(max;price) fby ([]sym;exch)
sym exch price size
-------------------
A   L    20    100
A   N    21    101
B   L    46    102
```


 email:info@aquaq.co.uk

Where Clause

- If you require a multivalent function in fby, you can usually restructure the required function to be monadic
- A good approach is to use a table argument
- In the following example we want to extract all the trades with price greater than VWAP. The VWAP calculation is diadic.

email:info@aquaq.co.uk

Where Clause

```
// This is what we want
q)t1:delete vwap from select from
   (update vwap:size wavg price by sym from trades) where price>vwap
q)t1
time                          sym  src price size
-------------------------------------------------
2014.03.12D08:00:22.311000000 IBM  L   43.54 392
2014.03.12D08:00:27.689000000 IBM  L   43.56 1504
2014.03.12D08:00:27.906000000 YHOO N   35.55 4404
2014.03.12D08:00:30.579000000 IBM  L   43.57 470
2014.03.12D08:00:43.114000000 YHOO L   35.52 5113
..
// fby version
q)select from trades
   where price>({x[`size] wavg x`price};([]size;price)) fby sym
time                          sym  src price size
-------------------------------------------------
2014.03.12D08:00:22.311000000 IBM  L   43.54 392
2014.03.12D08:00:27.689000000 IBM  L   43.56 1504
2014.03.12D08:00:27.906000000 YHOO N   35.55 4404
2014.03.12D08:00:30.579000000 IBM  L   43.57 470
2014.03.12D08:00:43.114000000 YHOO L   35.52 5113
..
// check they are the same
q)t1~select from trades
         where price>({x[`size] wavg x`price};([]size;price)) fby sym
1b
```


email:info@aquaq.co.uk

Data Modification

- When writing complex select statements, it is best to break it down to simple list examples
- Assume we have a table of trades, in time order
- You want to add a column to show, for every trade, whether it is an up tick (price higher than previous for the same instrument), down tick (lower than previous) or same tick (price the same as the previous)

email:info@aquaq.co.uk

Data Modification

- The data looks like this (loaded from fakedb.q):

```
time                          sym  src price size
-------------------------------------------------
2014.03.11D08:00:00.177000000 MSFT L   36.01 1427
2014.03.11D08:00:04.569000000 MSFT O   36.01 708
2014.03.11D08:00:09.230000000 CSCO N   35.5  7810
2014.03.11D08:00:20.322000000 ORCL N   32.17 1400
2014.03.11D08:00:22.311000000 IBM  L   43.54 392
2014.03.11D08:00:25.426000000 CSCO N   35.44 1935
2014.03.11D08:00:27.511000000 NOK  O   31.77 1600
2014.03.11D08:00:27.689000000 IBM  L   43.56 1504
2014.03.11D08:00:27.906000000 YHOO N   35.55 4404
2014.03.11D08:00:30.278000000 AAPL N   25.37 6889
2014.03.11D08:00:30.579000000 IBM  L   43.57 470
..
```


email:info@aquaq.co.uk

Data Modification

- One difficulty is that all the trades for different instruments are mixed together
- However, we can ignore this as kdb+ will handle it in the by clause
- To simplify the problem we consider the case of a simple list. We want to create a uniform function which produces 1 if the value is greater than the previous, 0 if it is the same, and -1 if it is less than the previous.
- We can do this with a combination of keywords (we do not want to use loops!)

```
q)prices:3 2 2 1 5
q)deltas prices
3 -1 0 -1 4
q)signum deltas prices
1 -1 0 -1 1i
```


email:info@aquaq.co.uk

Data Modification

- To be precise, we probably want to drop the first value and make it a 0 (no move) value rather than an uptick
- And we want to create a function from it

```
q)tickdirection:{0i,1 _ signum deltas x}
q)tickdirection prices
0 -1 0 -1 1i
```


email:info@aquaq.co.uk

Data Modification

- We can then use this in the table. We want to add a column, so we need an update statement. To differentiate between different symbols, we will add a by clause.

```
q)update dir:tickdirection price by sym from trades
time                          sym  src price size dir
-----------------------------------------------------
2014.03.11D08:00:00.177000000 MSFT L   36.01 1427 0
2014.03.11D08:00:04.569000000 MSFT O   36.01 708  0
2014.03.11D08:00:09.230000000 CSCO N   35.5  7810 0
2014.03.11D08:00:20.322000000 ORCL N   32.17 1400 0
2014.03.11D08:00:22.311000000 IBM  L   43.54 392  0
2014.03.11D08:00:25.426000000 CSCO N   35.44 1935 -1
2014.03.11D08:00:27.511000000 NOK  O   31.77 1600 0
2014.03.11D08:00:27.689000000 IBM  L   43.56 1504 1
2014.03.11D08:00:27.906000000 YHOO N   35.55 4404 0
2014.03.11D08:00:30.278000000 AAPL N   25.37 6889 0
2014.03.11D08:00:30.579000000 IBM  L   43.57 470  1
2014.03.11D08:00:32.203000000 MSFT O   36.04 2522 1
2014.03.11D08:00:43.114000000 YHOO L   35.52 5113 -1
2014.03.11D08:00:47.140000000 YHOO L   35.49 447  -1
2014.03.11D08:00:56.679000000 AAPL O   25.38 6103 1
..
```


email:info@aquaq.co.uk

Data Modification

- We can then calculate the total size traded on upticks, downticks and no movements

```
q)select sum size by sym,dir from update dir:tickdirection price by sym
  from trades
sym  dir| size
--------| -------
AAPL -1 | 1297566
AAPL 0  | 410155
AAPL 1  | 1243134
CSCO -1 | 1284603
CSCO 0  | 389397
CSCO 1  | 1168089
..
```

- It may appear that we should be able to do this as:

```
q)select sum size by sym,dir:tickdirection price from trades
sym  dir| size
--------| -------
AAPL -1 | 2774632
AAPL 0  | 82138
AAPL 1  | 94085
CSCO -1 | 960982
..
```

Data Modification

- However, this is not the case as tickdirection is calculated on the price column as a whole rather than by splitting on sym first
- We can however use an fby:

```
q)select sum size by sym,dir:(tickdirection;price) fby sym from trades
sym  dir| size
--------| -------
AAPL -1 | 1297566
AAPL 0  | 410155
AAPL 1  | 1243134
CSCO -1 | 1284603
CSCO 0  | 389397
..
```


email:info@aquaq.co.uk

Data Modification

- To try: we want to calculate the total number of trades and the total volume traded within a ‘price group’.
  A price group is defined as a set of trades which occur consecutively at the same price for a given instrument.

```
q)show example
time  price size
----------------
09:00 10    100
09:01 11    200
09:02 11    150
09:03 12    400
09:04 12    300
09:05 11    600
```

- In the above example, there are 4 price groups:
  100@10, 350@11, 700@12, 600@11

email:info@aquaq.co.uk

Data Modification

```
q)price:10 11 11 12 12 11
q)differ price
110101b
q)sums differ price
1 2 2 3 3 4i
q)select sum size, count i by pricegroup:sums differ price from example
pricegroup| size x
----------| ------
1         | 100  1
2         | 350  2
3         | 700  2
4         | 600  1
// If there was multiple symbols the solution would need to be similar
// to the previous example, i.e.
// select sum size, count i
// by sym, pricegroup:({sums differ x};price) fby sym
// from trades
```

Grouping

- A lot of analysis involves doing simple statistical operations (max, min, first, last, avg, count etc.) but doing more complex grouping operations
- Any function can be used to group data. Grouping functions are usually uniform
- As an example consider xbar:

```
q)select counter:count i by time:45 xbar time.minute from trades
time | counter
-----| -------
07:30| 273
08:15| 912
09:00| 908
09:45| 866
10:30| 895
..
```

- Our data only starts at 08:00, but the bars start at 07:30 as it is 45 minute buckets since midnight.
  How can we shift it?

 email:info@aquaq.co.uk

Grouping

```
q)select counter:count i by time:08:00+45 xbar time.minute - 08:00
  from trades
time | counter
-----| -------
08:00| 875
08:45| 917
09:30| 886
10:15| 870
11:00| 860
..
// can create a function for it
q)timeshiftbar:{[start;minbar;times] start+minbar xbar (`minute$times)-start}
q)select counter:count i by time:timeshiftbar[08:00;45;time] from trades
time | counter
-----| -------
08:00| 875
08:45| 917
09:30| 886
10:15| 870
11:00| 860
..
```

Grouping

- Assume you want to group the data into different trade size buckets
- Trades with size 0 to 1000 are small, greater than 9000 are large, and everything in between is medium
- To do this, we create a function to map a vector of trades sizes to a set of indicators
- Again we go back to simple list examples

```
q)sizes:2400 98 5677 9800
// use bin to find correct position
q)0 1000 9000 bin sizes
1 0 1 2
// map it to keywords
q)`small`med`big 0 1000 9000 bin sizes
`med`small`med`big
// alternatively we can use a dictionary
// this relies on the value of the dict being sorted
q)d:`small`med`big!0 1000 9000
q)d bin sizes
`med`small`med`big
```


email:info@aquaq.co.uk

Grouping

```
// can drop this into a select statement
q)select count i
  by sizebucket:(`small`med`big!0 1000 9000) bin size from trades
sizebucket| x
----------| ----
big       | 73
med       | 6808
small     | 3119

// or grouped by sym
q)select count i
  by sym,sizebucket:{(`small`med`big!0 1000 9000) bin x} fby sym
  from trades
sym  sizebucket| x
---------------| ---
AAPL big       | 7
AAPL med       | 795
AAPL small     | 318
CSCO big       | 2
CSCO med       | 773
..
```


email:info@aquaq.co.uk

Grouping

- We would like to calculate the number of trades which occur in different areas of the trading range.
  For example, if a stock trades between 100 and 105 on a given day, the trading range is 5. The bottom 25% are the trades between 100 and 101.25, and the top 25% are the trades between 103.75 and 105.

  Create a function to map a list of prices into one of `bot`mid`top and use in a select statement to calculate the count and total size traded in the range.
  Assume the list of trades is large. You should consider using iasc or rank in the function.

email:info@aquaq.co.uk

Grouping

```
// create price list
q)prices:100+100?5f
q)prices
101.6946 102.5957 100.3766 100.6854 101.8615 103.0312..
// work out the points that you need to index at
q)`int$0 .25 .75 * count prices
0 25 75i
// get the positions and prices which indicate the breaks
q)(iasc prices)`int$0 .25 .75 * count prices
80 0 50
q)prices (iasc prices)`int$0 .25 .75 * count prices
100.1004 101.6946 103.9027
q)(`bot`mid`top!prices (iasc prices)`int$0 .25 .75 * count prices) bin prices
`mid`mid`bot`bot`mid`mid`top`bot`mid`top`bot`bot`bot..
// create a function
q)bmt:{(`bot`mid`top!x (iasc x)`int$0 .25 .75 * count x) bin x}
q)select count i, sum size by sym,BMT:(bmt;price)fby sym from trades
sym  BMT| x   size
--------| -----------
AAPL bot| 261 726452
AAPL mid| 574 1474979
AAPL top| 285 749424
CSCO bot| 280 697888
CSCO mid| 568 1420370
..
```

Racking

- A common operation is to aggregate and downsample data using xbar. xbar only produces buckets where there is a value
- If a contiguous timeseries is required, then a 'rack' must be produced to join against

```
q)show t1
sym time  size price
--------------------
b   09:00 200  30.9
a   09:01 100  36
b   09:16 1200 30.9
a   09:32 200  36.1
a   09:34 400  36.2

// not all minute buckets are populated
q)select sum size by sym,15 xbar time.minute from t1
sym minute| size
----------| ----
a   09:00 | 100
a   09:30 | 600
b   09:00 | 200
b   09:15 | 1200
```


email:info@aquaq.co.uk

Racking

```
// define a start and end time
q)start:09:00
q)end:09:59

// create a contiguous list of times
q)times:start + bucketsize*til 1+floor(end-start)%bucketsize
q)times
09:00 09:15 09:30 09:45

// build a rack from it
q)rack:(`sym xasc select distinct sym from t1) cross ([]minute:times)
q)rack
sym minute
----------
a   09:00
a   09:15
a   09:30
a   09:45
b   09:00
b   09:15
b   09:30
b   09:45
```


email:info@aquaq.co.uk

Racking

```
// join the source data to the rack
q)rack lj select sum size by sym,15 xbar time.minute from t1
sym minute size
---------------
a   09:00  100
a   09:15
a   09:30  600
a   09:45
b   09:00  200
b   09:15  1200
b   09:30
b   09:45

// or like this
q)rack#select sum size by sym,15 xbar time.minute from t1
sym minute| size
----------| ----
a   09:00 | 100
a   09:15 |
a   09:30 | 600
a   09:45 |
b   09:00 | 200
b   09:15 | 1200
b   09:30 |
b   09:45 |
```


email:info@aquaq.co.uk

Racking

```
// Some times we need to then fill in the gaps with a value
q)0^rack#select sum size by sym,15 xbar time.minute from t1
sym minute| size
----------| ----
a   09:00 | 100
a   09:15 | 0
a   09:30 | 600
a   09:45 | 0
b   09:00 | 200
b   09:15 | 1200
b   09:30 | 0
b   09:45 | 0

// Or sometimes fill forward values
q)update fills price by sym from
   update 0^size from
   rack#select sum size,last price by sym,15 xbar time.minute from t1
sym minute| size price
----------| ----------
a   09:00 | 100  36
a   09:15 | 0    36
a   09:30 | 600  36.2
a   09:45 | 0    36.2
b   09:00 | 200  30.9
b   09:15 | 1200 30.9
b   09:30 | 0    30.9
b   09:45 | 0    30.9
```


 email:info@aquaq.co.uk

Aligning Timeseries

- A common operation is to align two sets of asynchronous timeseries data
- Timestamps rarely match
- Alignment can be for different datasets for the same instrument (e.g. trades and quotes) or across instruments (align GOOG trades with IBM trades)
- There are three main approaches
  1. **bucketing**: the data sets are down-sampled to a common set of time points and joined using an lj. This is based on the 'racking' approach explained previously.
  2. **asof**: one set is the master set, and the secondary set is aligned to it
  3. **full**: all data at every time point

email:info@aquaq.co.uk

Aligning Timeseries

- The first we will look at is **asof**
- kdb+ has an inbuilt join, aj, which can be used for this
- The signature of aj is
  aj[(columns to do exact match on),asof column; source table; value table]
- For performance and accuracy reasons, it is recommended that:
  - There is at most 1 exact match column
  - The value table should have an attribute (p or g) on the exact match column, and be sorted by the asof column within each exact match group
- It should be noted that the asof column can be any numeric column - it does not have to be a time field

email:info@aquaq.co.uk

Aligning Timeseries

```
q)trades
time                          sym  src price size
--------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:04.569000000 MSFT O   36.01 708
..
// check the attribute on sym
q)meta quotes
c   | t f a
----| -----
time| p
sym | s   g
src | s   g
..
// align every trade with the prevailing quote
q)aj[`sym`time;trades;quotes]
time                          sym  src price size bid   ask   bsize asize
--------------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:04.569000000 MSFT O   36.01 708  36    36.01 5000  1500
2014.03.14D08:00:09.230000000 CSCO N   35.5  7810 35.49 35.5  10000 10000
2014.03.14D08:00:20.322000000 ORCL N   32.17 1400 32.17 32.19 3500  1000
2014.03.14D08:00:22.311000000 IBM  L   43.54 392  43.54 43.56 5500  4500
..
```


email:info@aquaq.co.uk

Aligning Timeseries

```
// Can also use aj to align prices from different instruments
// In this case, the prevailing IBM price when GOOG trades
q)t1:select time,GOOGprice:price from trades where sym=`GOOG
q)t2:select time,IBMprice:price from trades where sym=`IBM
q)aj[`time;t1;t2]
time                          GOOGprice IBMprice
------------------------------------------------
2014.03.14D08:01:23.868000000 41.37     43.54
2014.03.14D08:01:31.205000000 41.33     43.54
2014.03.14D08:01:31.782000000 41.3      43.54
2014.03.14D08:01:59.375000000 41.24     43.48
2014.03.14D08:02:40.957000000 41.33     43.49
..
```


email:info@aquaq.co.uk

Aligning Timeseries

- There are different approaches to creating a 'full' data set, using either a uj or an aj
- A full data set is essentially all data interleaved, rather than one set at the time of another
- The first approach we will consider is using **uj** (union join) to create the superset of data, then sorting and filling it
- The second approach involves using an **aj** against the superset of times. The aj approach can be faster, but is less flexible

email:info@aquaq.co.uk

Aligning Timeseries

```
q)t:`time xasc trades uj quotes
time                          sym  src price size bid   ask   bsize asize
-------------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:00.456000000 MSFT L              36.02 36.06 3000  500
2014.03.14D08:00:00.957000000 MSFT N              36.03 36.04 8500  4000
2014.03.14D08:00:01.387000000 MSFT O              36    36.03 6500  7000
2014.03.14D08:00:01.548000000 ORCL L              32.2  32.23 3000  3500
2014.03.14D08:00:02.009000000 MSFT N              36    36.02 4500  7000
2014.03.14D08:00:03.847000000 IBM  N              43.54 43.56 4000  9000
2014.03.14D08:00:03.859000000 MSFT O              36    36.01 5000  1500
..
// usually need to fill data forward
q)update fills price, fills size, fills bid, fills ask, fills bsize,
 fills asize by sym from t
time                          sym  src price size bid   ask   bsize asize
-------------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:00.456000000 MSFT L   36.01 1427 36.02 36.06 3000  500
2014.03.14D08:00:00.957000000 MSFT N   36.01 1427 36.03 36.04 8500  4000
2014.03.14D08:00:01.387000000 MSFT O   36.01 1427 36    36.03 6500  7000
2014.03.14D08:00:01.548000000 ORCL L              32.2  32.23 3000  3500
2014.03.14D08:00:02.009000000 MSFT N   36.01 1427 36    36.02 4500  7000
2014.03.14D08:00:03.847000000 IBM  N              43.54 43.56 4000  9000
2014.03.14D08:00:03.859000000 MSFT O   36.01 1427 36    36.01 5000  1500
..
```


email:info@aquaq.co.uk

Aligning Timeseries

- The same alignments can be done with an aj
- The aj is usually faster but restricts to always filling forward
- If 'sampling' data (e.g. get the prevailing value every minute) then building a rack and aj-ing against is usually faster than the equivalent xbar statement (select last price, last size ... by sym, 1 xbar time.minute from trade)

email:info@aquaq.co.uk

Aligning Timeseries

```
// build the full series and aj both tables against it
q)fullseries:`time xasc (select time,sym from trades),
   select time,sym from quotes
q)aj[`sym`time;aj[`sym`time;fullseries;trades];quotes]
time                          sym  src price size bid   ask   bsize asize
-------------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:00.456000000 MSFT L   36.01 1427 36.02 36.06 3000  500
2014.03.14D08:00:00.957000000 MSFT N   36.01 1427 36.03 36.04 8500  4000
2014.03.14D08:00:01.387000000 MSFT O   36.01 1427 36    36.03 6500  7000
2014.03.14D08:00:01.548000000 ORCL L              32.2  32.23 3000  3500
2014.03.14D08:00:02.009000000 MSFT N   36.01 1427 36    36.02 4500  7000
2014.03.14D08:00:03.847000000 IBM  N              43.54 43.56 4000  9000
2014.03.14D08:00:03.859000000 MSFT O   36.01 1427 36    36.01 5000  1500
// or equivalently (and more extensibly) with an adverb
q)aj[`sym`time]/[fullseries;(trades;quotes)]
time                          sym  src price size bid   ask   bsize asize
-------------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427
2014.03.14D08:00:00.456000000 MSFT L   36.01 1427 36.02 36.06 3000  500
2014.03.14D08:00:00.957000000 MSFT N   36.01 1427 36.03 36.04 8500  4000
2014.03.14D08:00:01.387000000 MSFT O   36.01 1427 36    36.03 6500  7000
2014.03.14D08:00:01.548000000 ORCL L              32.2  32.23 3000  3500
2014.03.14D08:00:02.009000000 MSFT N   36.01 1427 36    36.02 4500  7000
2014.03.14D08:00:03.847000000 IBM  N              43.54 43.56 4000  9000
2014.03.14D08:00:03.859000000 MSFT O   36.01 1427 36    36.01 5000  1500
```

Trade Analysis Example

- An aj can be used in conduction with a shifted timeseries to calculate the prevailing value at a given point in time +/- T
- An example of this would be to calculate, for every trade, how much the price moves within the 5 minutes after the trade

email:info@aquaq.co.uk

Trade Analysis Example

```
// add a column containing the price in 5 minutes for every trade
// by shifting the time column back by 5 minutes
q)aj[`sym`time;trades;
  select sym,time:time - 0D00:05, priceplus5:price from trades]
time                          sym  src price size priceplus5
------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427 35.95
2014.03.14D08:00:04.569000000 MSFT O   36.01 708  35.95
2014.03.14D08:00:09.230000000 CSCO N   35.5  7810 35.78
2014.03.14D08:00:20.322000000 ORCL N   32.17 1400 32.19
2014.03.14D08:00:22.311000000 IBM  L   43.54 392  43.49
..

// calculate the pnl for the buyer for every trade after 5 minutes
q)update pnl:size*priceplus5 - price
from aj[`sym`time;trades;
select sym,time:time - 0D00:05, priceplus5:price from trades]
time                          sym  src price size priceplus5 pnl
-------------------------------------------------------------------
2014.03.14D08:00:00.177000000 MSFT L   36.01 1427 35.95      -85.62
2014.03.14D08:00:04.569000000 MSFT O   36.01 708  35.95      -42.48
2014.03.14D08:00:09.230000000 CSCO N   35.5  7810 35.78      2186.8
2014.03.14D08:00:20.322000000 ORCL N   32.17 1400 32.19      28
2014.03.14D08:00:22.311000000 IBM  L   43.54 392  43.49      -19.6
..
```

Trade Analysis Example

```
// we can generise this to a very powerful function.
// First we create a function to shift the time and rename the price
q)sel:{(`sym`second,`$$[x<0;"MINUS";"PLUS"],string abs x)xcol
select sym,time.second-x,price from trades}
q)sel 1
sym  second   PLUS1
-------------------
MSFT 07:59:59 36.01
MSFT 08:00:03 36.01
CSCO 08:00:08 35.5
ORCL 08:00:19 32.17
IBM  08:00:21 43.54
..
q)sel -1
sym  second   MINUS1
--------------------
MSFT 08:00:01 36.01
MSFT 08:00:05 36.01
CSCO 08:00:10 35.5
ORCL 08:00:21 32.17
IBM  08:00:23 43.54
..
```


email:info@aquaq.co.uk

Trade Analysis Example

```
// Test we can use the new version in the same way as the previous
q)aj[`sym`second;select sym,time.second,price,size from trades;sel 300]
sym  second   price size PLUS300
--------------------------------
MSFT 08:00:00 36.01 1427 35.95
MSFT 08:00:04 36.01 708  35.95
CSCO 08:00:09 35.5  7810 35.78
ORCL 08:00:20 32.17 1400 32.19
IBM  08:00:22 43.54 392  43.49
..

// we can tie this in with an adverb to give the ability to get the traded
// price at any +/- time point relative to the trade
q)priceat:
  {{aj[`sym`second;x;sel y]}/[select sym,time.second,price,size from x;y]}
q)priceat[trades;-120 -1 0 5 60 300]
sym  second   price size MINUS120 MINUS1 PLUS0 PLUS5 PLUS60
-----------------------------------------------------------------
MSFT 08:00:00 36.01 1427                 36.01 36.01 36.04
MSFT 08:00:04 36.01 708           36.01  36.01 36.01 36.04
CSCO 08:00:09 35.5  7810                 35.5  35.5  35.44
ORCL 08:00:20 32.17 1400                 32.17 32.17 32.18
IBM  08:00:22 43.54 392                  43.54 43.56 43.54
```

  email:info@aquaq.co.uk

Trade Analysis Example

```
// we can then calculate the pnl at each time point
// drop the fixed columns and flip t a dictionary
q)flip `sym`second`price`size _ t:priceat[trades;-120 -1 0 5 60]
MINUS120|                                                               ..
MINUS1  |       36.01                   35.5        43.54               ..
PLUS0   | 36.01 36.01 35.5  32.17 43.54 35.44 31.77 43.56 35.55 25.37 ..
PLUS5   | 36.01 36.01 35.5  32.17 43.56 35.44 31.77 43.57 35.55 25.37 ..
PLUS60  | 36.04 36.04 35.44 32.18 43.54 35.61 31.77 43.54 35.49 25.38 ..

// subtract the price from each column
q)(flip `sym`second`price`size _ t) -\: t`price
MINUS120|                                                               ..
MINUS1  |       0                    0.06       -0.02            -0...
PLUS0   | 0     0     0     0     0     0     0     0     0     0     0   ..
PLUS5   | 0     0     0     0     0.02  0     0     0.01  0     0     0   ..
PLUS60  | 0.03  0.03  -0.06 0.01  0     0.17  0     -0.02 -0.06 0.01  -0...

// multiply by size, flip back to a table, and join on the standard columns
q)(`sym`second`price`size#t),'
   flip t[`size]*/:(flip `sym`second`price`size _ t) -\: t`price
sym  second   price size MINUS120 MINUS1  PLUS0 PLUS5   PLUS60
-----------------------------------------------------------------------
MSFT 08:00:00 36.01 1427                  0     0       42.81
MSFT 08:00:04 36.01 708           0       0     0       21.24
CSCO 08:00:09 35.5  7810                  0     0       -468.6
ORCL 08:00:20 32.17 1400                  0     0       14
```

Trade Analysis Example

email:info@aquaq.co.uk

Trade Analysis Example

```
// we can create a pnl function and use it convert the table to PnLs
q)pnl:{(`sym`second`price#x),'
  flip x[`size]*/:(flip `sym`second`price _ x) -\: x`price}
q)pnl priceat[trades;-120 -1 0 5 60]
sym  second   price size         MINUS120 MINUS1  PLUS0 PLUS5   PLUS60
---------------------------------------------------------------------
MSFT 08:00:00 36.01 1984943                       0     0       42.81
MSFT 08:00:04 36.01 475768.9              0       0     0       21.24
CSCO 08:00:09 35.5  6.071884e+07                  0     0       -468.6
ORCL 08:00:20 32.17 1914962                       0     0       14
IBM  08:00:22 43.54 136596.3                      0     7.84    0

// If we apply a key to the table, we can simply sum all the columns
// to get total PnL for every input
q)sum 4!pnl priceat[trades;-1 0 5 60]
MINUS1  | -9210.05
PLUS0   | -311.28
PLUS5   | -993.5
PLUS60  | 23071.98

// and test performance
q)count trades
10000
q)\ts sum 4!pnl priceat[trades;120 300 6000]
6 1639488
q)\ts sum 4!pnl priceat[trades;1 5 60 120 300 600 1200 60000]
14 3605888
```

Trade Analysis Example

- So with 3 functions we can calculate the PnL at different time points around every trade... and it's fast.

```
sel:{(`sym`second,`$$[x<0;"MINUS";"PLUS"],string abs x)xcol
 select sym,time.second-x,price from trades}
priceat:{{aj[`sym`second;x;sel y]}/
 [select sym,time.second,price,size from x;y]}
pnl:{(`sym`second`price#x),'flip x[`size]*/:
 (flip `sym`second`price _ x) -\: x`price}
```

- We can also do other slice and dice analysis as normal e.g.

```
q)select sum PLUS120,sum PLUS300 by sym from pnl priceat[trades;120 300]
sym | PLUS120   PLUS300
----| -------------------
AAPL| 19052.79  25615.99
CSCO| 17946.22  31186.06
DELL| 8451.93   33683.34
GOOG| -22859.06 -74431.08
IBM | -1024.36  21337.97
MSFT| 9486.47   8767.13
NOK | 3012.95   23461.22
ORCL| 27393.33  60304.42
YHOO| -3151.27  -53086.71
```

Offset Alignment

- The previous example is primarily an example of offset alignment - shifting the time field of one table to calculate something either side of another
- Using a similar approach, calculate for every trade the VWAP for the 5 minutes following the trade (excluding the contribution from the current trade)
- Remember that VWAP can be calculated using the running sum of the size and the running sum of size*price

email:info@aquaq.co.uk

Offset Alignment

```
// add in the running sum of price*size and size
q)show trades:update sumps:sums price*size, sumsize:sums size
 by sym from trades
time                          sym  src price size sumps    sumsize
-----------------------------------------------------------------
2014.03.17D08:00:00.177000000 MSFT L   36.01 1427 51386.27 1427
2014.03.17D08:00:04.569000000 MSFT O   36.01 708  76881.35 2135
2014.03.17D08:00:09.230000000 CSCO N   35.5  7810 277255   7810
2014.03.17D08:00:20.322000000 ORCL N   32.17 1400 45038    1400

// shift the time, join on the values
q)t:aj[`sym`time;trades;
 select time-0D00:05,sym,sumps5:sumps,sumsize5:sumsize from trades]

// calculate the running VWAP from the difference
q)update vwap5:(sumps5 - sumps)%sumsize5-sumsize from delete time from t
sym  src price size sumps    sumsize sumps5  sumsize5 vwap5
-------------------------------------------------------------
MSFT L   36.01 1427 51386.27 1427    1276302 35462    35.98987
MSFT O   36.01 708  76881.35 2135    1276302 35462    35.98944
CSCO N   35.5  7810 277255   7810    1296364 36364    35.69058
ORCL N   32.17 1400 45038    1400    1081146 33589    32.18826
```

Pivoting

- Data can be pivoted to move row values into column values
- A generic pivot function is available on code.kx

```
q)t1:select last price by sym,2 xbar time.second from trades
q)show t1
sym  second  | price
-------------| -----
AAPL 08:00:30| 25.37
AAPL 08:00:56| 25.38
AAPL 08:01:20| 25.38
AAPL 08:01:32| 25.38
AAPL 08:01:52| 25.39
..
```


email:info@aquaq.co.uk

Pivoting

```
// extract the distinct list of column headers
q)H:asc exec distinct sym from t1

// extract dictionaries of sym|price at every time point
q)exec sym!price by second from t1
08:00:00| (,`MSFT)!,36.01
08:00:04| (,`MSFT)!,36.01
08:00:08| (,`CSCO)!,35.5
08:00:20| (,`ORCL)!,32.17
08:00:22| (,`IBM)!,43.54
08:00:24| (,`CSCO)!,35.44
08:00:26| `IBM`NOK`YHOO!43.56 31.77 35.55
..

// make each dictionary fully populated
q)exec H#sym!price by second from t1
        | AAPL CSCO DELL GOOG IBM   MSFT  NOK ORCL  YHOO
--------| ----------------------------------------------
08:00:00|                           36.01
08:00:04|                           36.01
08:00:08|      35.5
08:00:20|                                     32.17
08:00:22|                     43.54
..
```

Pivoting

```
// and add a header to the key column
q)exec H#sym!price by second:second from t1
second  | AAPL CSCO DELL GOOG IBM   MSFT  NOK ORCL  YHOO
--------| ----------------------------------------------
08:00:00|                           36.01
08:00:04|                           36.01
08:00:08|      35.5
08:00:20|                                     32.17
08:00:22|                     43.54
..

// can fill forward if required
q)fills exec H#sym!price by second:second from t1
second  | AAPL CSCO DELL GOOG IBM   MSFT  NOK ORCL  YHOO
--------| ----------------------------------------------
08:00:00|                           36.01
08:00:04|                           36.01
08:00:08|      35.5                 36.01
08:00:20|      35.5                 36.01     32.17
08:00:22|      35.5           43.54 36.01     32.17
..
```

Functional Form

- Statements can be built and executed in functional form
- Functional form can be seen by using parse on statement
- Functional form can be used to build column, by and where clauses dynamically
- It offers no performance advantage
- It should be only be used when some part of the statement is dynamic e.g. the column names, the required aggregations, the grouping or the condition criteria

```
q)parse"select sum size,max price by sym from trades
      where time.time>09:00"
?
`trades
,,(>;`time.time;09:00)
(,`sym)!,`sym
`size`price!((sum;`size);(max;`price))
```


email:info@aquaq.co.uk

Functional Form

- It is sometimes possible to implement dynamic behaviour without functional form e.g. selecting columns can done by building a list of column names and using #
- However, sometimes it is the simplest approach. Consider as an example a function to calculate the max price and total size traded from the trade table based on different groupings
- Functional form is a very powerful tool, but it should only be used when appropriate as it leads to harder to understand code

email:info@aquaq.co.uk

Functional Form

```
// Parse the statement to get the template
q)parse"select sum size,max price by A,B from trades"
?
`trades
()
`A`B!`A`B
`size`price!((sum;`size);(max;`price))
// create a function
q)grptrade:
     {eval(?;`trades;();x!x,:();`size`price!((sum;`size);(max;`price)))}
// run it for different groupings
q)grptrade[`sym]
sym | size    price
----| -------------
AAPL| 2950855 26.49
CSCO| 2842089 36.74
DELL| 2792637 30.44
GOOG| 2761650 41.59
q)grptrade[`sym`src]
sym  src| size    price
--------| -------------
AAPL L  | 981671  26.46
AAPL N  | 935468  26.49
AAPL O  | 1033716 26.47
CSCO L  | 906250  36.74
..
```

Trade Analysis Example - Revisited

- Recall we created a function to calculate PnL at different time points

```
q)pnl priceat[trades;120 300]
sym  second   price size          PLUS120 PLUS300
--------------------------------------------------
MSFT 08:00:00 36.01 1984943       -57.08  -85.62
MSFT 08:00:04 36.01 475768.9      -28.32  -42.48
CSCO 08:00:09 35.5  6.071884e+07 859.1   2186.8
ORCL 08:00:20 32.17 1914962       -42     28
IBM  08:00:22 43.54 136596.3      -23.52  -19.6
```

- We can extend this with functional form to calculate different statistics at each time point, grouped by different fields
- In this case, the grouping columns are dynamic, and the columns to group are dynamic as it depends on the user input of the time points

email:info@aquaq.co.uk

Trade Analysis Example - Revisited

```
// get the template
q)parse"select sum PLUS120 by sym from trade"
?
`trade
()
(,`sym)!,`sym
(,`PLUS120)!,(sum;`PLUS120)

// and some sample data
q)cols x:pnl priceat[trades;120 300]
`sym`second`price`size`PLUS120`PLUS300

// the column dict can be built like this
// match on column names like "PLUS*" or "MINUS*"
q)c where any (c:cols x) like/:("PLUS*";"MINUS*")
`PLUS120`PLUS300
q)c!sum,/:c:c where any (c:cols x) like/:("PLUS*";"MINUS*")
PLUS120| sum `PLUS120
PLUS300| sum `PLUS300

// create a function to aggregate by whatever columns are supplied
q)aggpnl:{[g;x] eval(?;x;();g!g,:();
 c!sum,/:c:c where any (c:cols x) like/:("PLUS*";"MINUS*"))}
```


email:info@aquaq.co.uk

Trade Analysis Example - Revisited

```
// run it
q)aggpnl[`sym] pnl priceat[trades;120 300]
sym | PLUS120   PLUS300
----| -------------------
AAPL| 19052.79  25615.99
CSCO| 17946.22  31186.06
DELL| 8451.93   33683.34
GOOG| -22859.06 -74431.08
IBM | -1024.36  21337.97

// we can use previous techniques to add other fields to aggregate on
q)aggpnl[`sym`TOD]
update TOD:`1morning`2afternoon`3late 00:00:00 10:00:00 15:00:00 bin second
from  pnl priceat[trades;-60 120 300 900]
sym  TOD       | MINUS60  PLUS120   PLUS300   PLUS900
---------------| --------------------------------------
AAPL 1morning  | -398.33  6636.01   15259.14  31933.25
AAPL 2afternoon| 3607.82  9152.86   4924.81   24421.47
AAPL 3late     | 4482.92  3263.92   5432.04   -139.97
CSCO 1morning  | -3130.13 7082.43   13852.56  11162.97
CSCO 2afternoon| 2932.05  12376.91  18785.82  49853.54
CSCO 3late     | 5413.5   -1513.12  -1452.32  -13242.19
```

Book Analysis

- Imagine we have a table of depth information, with a wide table containing price and size information for different levels of depth (this is similar to how Reuters distribute depth data)

```
q)d:select time,sym,bid1,bsize1,bid2,bsize2,bid3,bsize3
 from depth where sym=`GOOG
q)d
sym  bid1  bsize1 bid2  bsize2 bid3  bsize3
-------------------------------------------
GOOG 41.3  5500   41.29 8000   41.28 6500
GOOG 41.3  5500   41.29 6500   41.28 8000
GOOG 41.3  5500   41.29 7000   41.28 7500
GOOG 41.3  5500   41.29 6000   41.28 10000
GOOG 41.3  4500   41.29 7000   41.28 7500
GOOG 41.3  4500   41.29 5000   41.28 9500
```

- Assume we have an order of size 10000 to fill. We would like to calcuate, at every time point, the VWAP we will achieve on the bid

Book Analysis

- We could iterate through every row of the table ...
- ... or we could do the whole table at once
- To do the whole table at once, we can work on the columns individually

```
// extract the prices and sizes
// as test data to build the function
q)b1:d`bid1; b2:d`bid2; b3:d`bid3
q)s1:d`bsize1; s2:d`bsize2; s3:d`bsize3

// show the sizes
q)(s1;s2;s3)
5500 5500 5500 5500  4500 4500 10000 1500 1500 1500 3500 3500 3500 350..
8000 6500 7000 6000  7000 5000 12500 2000 3000 2000 4500 6000 4000 400..
6500 8000 7500 10000 7500 9500 12000 5000 4000 6000 4500 6500 8500 800..
```


email:info@aquaq.co.uk

Book Analysis

```
// compute the running sum of the sizes
q)sums (s1;s2;s3)
5500  5500  5500  5500  4500  4500  10000 1500 1500 1500 3500  3500  3..
13500 12000 12500 11500 11500 9500  22500 3500 4500 3500 8000  9500  7..
20000 20000 20000 21500 19000 19000 34500 8500 8500 9500 12500 16000 1..

// limit the timeseries at the order size
q)10000&sums (s1;s2;s3)
5500  5500  5500  5500  4500  4500  10000 1500 1500 1500 3500  3500  3..
10000 10000 10000 10000 10000 9500  10000 3500 4500 3500 8000  9500  7..
10000 10000 10000 10000 10000 10000 10000 8500 8500 9500 10000 10000 1..

// calculate the differences
// this gives the size traded at each price to fill the order
q)deltas 10000&sums (s1;s2;s3)
5500 5500 5500 5500 4500 4500 10000 1500 1500 1500 3500 3500 3500 3500..
4500 4500 4500 4500 5500 5000 0     2000 3000 2000 4500 6000 4000 4000..
0    0    0    0    0    500  0     5000 4000 6000 2000 500  2500 2500..

// calculate the wavg against the price
q)(deltas 10000&sums (s1;s2;s3)) wavg (b1;b2;b3)
41.2955 41.2955 41.2955 41.2955 41.2945 41.294 41.31 41.28588 41.28706..
```


 email:info@aquaq.co.uk

Book Analysis

```
// create a function from it
q)ordervwap:{[p;s;o] (deltas o&sums s) wavg p}
q)ordervwap[(b1;b2;b3); (s1;s2;s3); 10000]
41.2955 41.2955 41.2955 41.2955 41.2945 41.294 41.31 41.28588 41.28706..

// drop it into an update statement
q)update ovwap:ordervwap[(bid1;bid2;bid3);(bsize1;bsize2;bsize3);10000]
 from d
sym  bid1  bsize1 bid2  bsize2 bid3  bsize3 ovwap
----------------------------------------------------
GOOG 41.3  5500   41.29 8000   41.28 6500   41.2955
GOOG 41.3  5500   41.29 6500   41.28 8000   41.2955
GOOG 41.3  5500   41.29 7000   41.28 7500   41.2955
GOOG 41.3  5500   41.29 6000   41.28 10000  41.2955
GOOG 41.3  4500   41.29 7000   41.28 7500   41.2945
GOOG 41.3  4500   41.29 5000   41.28 9500   41.294
GOOG 41.31 10000  41.3  12500  41.29 12000  41.31

// time it
q)count depth
159800
q)\t update ovwap:ordervwap[(bid1;bid2;bid3);(bsize1;bsize2;bsize3);10000]
 from depth
8
```

Map-Reduce and Parallelisation

- When running queries against on disk, partitioned databases, kdb+ will automatically apply map-reduce algorithms to some operations
- These operations are run in each partition separately, then re-aggregated to give the final result
- The operations are:
  count, first, last, sum, prd, min, max, distinct, avg, wsum, wavg, var, dev, cov, cor, med

email:info@aquaq.co.uk

Map-Reduce and Parallelisation

```
q)select max price by sym from trade where date within 2013.05.01 2013.05.03
sym | price
----| -----
AAPL| 85.9
AIG | 27.64
..

// it works differently if aggregations are wrapped in lambdas
// or mixed with non-aggregations
q)select mp:{max x}[price],max price by sym
from trade where date within 2013.05.01 2013.05.03
sym | mp                  price
----| ------------------------------------
AAPL| 85.9  85.68 84.26 85.9  85.68 84.26
AIG | 27.64 27.62 26.79 27.64 27.62 26.79
..

// if an aggregation is first in the column list then
// any non-aggregations will cause errors
q)select max price,mp:{max x}[price] by sym
from trade where date within 2013.05.01 2013.05.03
'price
.
?
(+`sym`0!(`sym$`AAPL`AIG`AMD`DELL`DOW`GOOG`HPQ`IBM`INTC`MSFT`ORCL
```


 email:info@aquaq.co.uk

Map-Reduce and Parallelisation

- If the database is run with slaves, kdb+ will automatically parallelise and use a separate slave to do the processing separately in each partition

```
aquaq$ q hdb
KDB+ 3.1 2014.02.08 Copyright (C) 1993-2014 Kx Systems
q)\ts select size wavg price, max size by sym,15 xbar time.minute
   from trade
38 1777904

aquaq$ q hdb -s 2
KDB+ 3.1 2014.02.08 Copyright (C) 1993-2014 Kx Systems
q)\ts select size wavg price, max size by sym,15 xbar time.minute
   from trade
24 1183936
```

- In some cases the query can be restructured a little to make optimal use of slaves - do as much processing in the slave thread before returning the results to the main thread
- The following example calculates some statistics on the price movements for each stock.

 email:info@aquaq.co.uk

Map-Reduce and Parallelisation

```
q)\s
4
// calculate the max, avg, avg absolute and median price movement
// for a set of stocks in 15 minute buckets
// the slave threads are only selecting the data and running deltas
// all the stats are calculated in the main thread
q)\ts t1:select maxmove:max each price, avgmove:avg each price,
 avgabs:avg each abs each price,medmove:med each price by date,sym,minute
 from select 1 _ deltas price by sym,date,15 xbar time.minute from trade
73 7348544
q)t1
date       sym  minute| maxmove avgmove       avgabs     medmove
----------------------| --------------------------------------------
2013.05.01 AAPL 09:30 | 0.24    0.004814815   0.05641975 0.01
2013.05.01 AAPL 09:45 | 0.18    -0.004745763  0.04067797 -0.01
2013.05.01 AAPL 10:00 | 0.14    -0.0003921569 0.0454902  -0.01
// by contrast, in this example the slaves threads are calculating all
// the stats and returning the aggregated data to the main thread
q)\ts t2:select (max;avg;{avg abs x};med)@\:1 _ deltas price by
 sym,date,15 xbar time.minute from trade
26 1780864
q)t2
sym  date       minute| price
----------------------| ------------------------------------
AAPL 2013.05.01 09:30 | 0.24 0.004814815   0.05641975 0.01
AAPL 2013.05.01 09:45 | 0.18 -0.004745763  0.04067797 -0.01
AAPL 2013.05.01 10:00 | 0.14 -0.0003921569 0.0454902  -0.01
```


 email:info@aquaq.co.uk

Exercises: Queries

Please refer to kdbexercises.pdf

Chapter 15 contains further exercises on data analysis.

email:info@aquaq.co.uk

ETL

email:info@aquaq.co.uk

ETL

ETL: Extract, Transform, Load

- Data in character delimited format
- Text load function:

```
 (<column types>;<delimiter>) 0: <file handle>
/or
0:[ (<column types>;<delimiter>) ; <file handle> ]
```

where an enlisted delimiter assumes the row headers are the first line in the file and transforms directly into a table.

The parameters are as follows:

- **column types**

  refers to data type table
  character representation
  upper case
  '*' for nested character column
  ' ' to skip a field

ETL

- **delimiter**
  character that separates values
  a list that specifies width of columns
- **file handle**
  path to the file

- Data with column names on first row:

```
q)`:test.csv 0:("a,b";"1,1";"2,2")
`:test.csv
q)0:[("II";enlist",");`:test.csv]       /each column is loaded as rows
a b                                     /with integer types
---
1 1
2 2
```


 email:info@aquaq.co.uk

ETL

- Data without column names:

```
q)`:test2.csv 0:("1,1";"2,2";"3,3")
`:test2.csv
q)0:[("*S";",");`:test2.csv]          /each column is loaded as rows
,"1" ,"2" ,"3"
1    2    3
q)`a`b!0:[("*S";",");`:test2.csv]     /turn into dictionary
a| ,"1" ,"2" ,"3"                     /giving the proper column names
b| 1    2    3
q)flip `a`b!0:[("*S";",");`:test2.csv]  /flip into table
a    b                                /the "a" column elements are
------                                /read as string types
,"1" 1
,"2" 2
,"3" 3
```


email:info@aquaq.co.uk

ETL

- Data with fixed width:

```
q)`:test3.txt 0:(" 1  a  1 ";" 2  b  2 ";" 3  c  3 ")
`:test3.txt
q)0:[("J J";3 3 3);`:test3.txt]          /data read as long types
1 2 3
1 2 3
q)`a`b!0:[("J J";3 3 3);`:test3.txt]
a| 1 2 3
b| 1 2 3
q)flip `a`b!0:[("J J";3 3 3);`:test3.txt]       /flip into table
a b
---
1 1
2 2
3 3
```

- Use **read0** to read as plain text

```
q)read0`:test2.csv               /check whether it has column names
"1,1"
"2,2"
"3,3"
```

ETL

- Sometimes individual files are too large to read into memory at once. In these cases, files can be loaded and saved in chunks, using **.Q.fs** or **.Q.fsn**.
- These allow you to apply a function to the file, and read the file in chunks of data at a time - with .Q.fsn you can specify the size of the chunks in bytes.
- .Q.fs is a projection of .Q.fsn with the max ‘chunk’ size set to 131000 bytes.
- If we have a flat csv file containing trade data like this:

```
2014-04-14D08:00:00.093000000,NOK,N,31.71,1086
2014-04-14D08:00:00.210000000,ORCL,O,32.22,661
2014-04-14D08:00:00.243000000,IBM,L,43.53,2710
2014-04-14D08:00:01.276000000,CSCO,N,35.47,157
2014-04-14D08:00:01.779000000,YHOO,L,35.48,6883
2014-04-14D08:00:01.957000000,YHOO,L,35.53,310
...
```


 email:info@aquaq.co.uk

ETL

- We can apply a loading function which will insert to a table to that file using .Q.fs:

```
q)columns:`time`sym`src`price`size
q).Q.fs[{`tradetest insert flip columns!("PSSFI";",") 0:x}]`:trades.csv
                               |            |    |              |
                            make a tab     type  delim          file
```

- This results in a table as we would expect:

```
q)tradetest
time                          sym  src  price size
--------------------------------------------------
2014.04.14D08:00:00.093000000 NOK  N 31.71 1086
2014.04.14D08:00:00.210000000 ORCL O 32.22 661
2014.04.14D08:00:00.243000000 IBM  L 43.53 2710
2014.04.14D08:00:01.276000000 CSCO N 35.47 157
2014.04.14D08:00:01.779000000 YHOO L 35.48 6883
2014.04.14D08:00:01.957000000 YHOO L 35.53 310
...
```

ETL

- However, when the file is too large, it will not load to memory. In that case we can use .Q.fs to read in chunks and write these chunks directly to a table on disk instead of in memory:

```
q).Q.fs[{`:newfile upsert flip columns!("PSSFI";",") 0:x}]`:trades.csv
      |
     upsert to an on-disk table
```

- A tutorial on further usage of .Q.fs and some scripts illustrating usage of .Q.fsn are on code.kx.com:

  http://code.kx.com/wiki/Cookbook/LoadingFromLargeFiles

Exercises: ETL

Please refer to kdbexercises.pdf

Complete chapter 13 on ETL.

email:info@aquaq.co.uk

Idioms

email:info@aquaq.co.uk

Idioms

- change atom to list but keep list unchanged

```
q)type(),4
7h                           / 7h=converted to list
q)1 2 3~(),1 2 3             / (), keeps list unchanged
1b
q)count enlist 1 2 3         / enlist always adds a level of nesting
1
```

- fill from dictionary or don't change if null

```
q)d:1 2!3 4
q)v:1 10
q)v^d v
3 10
```

- Convert to symbol if not already one

```
q){$[10h=t:type x;`$x;-11h=t;x;`$string x]}
```

- Change one column in a splayed table without reading the table into memory

```
q)f set @[get f:`:trades/size;where get[`:trades/sym]=`IBM;*;10]
```

Big System Architecture

- Large scale kdb+ architectures consist of multiple processes with dedicated functions
- Processes can be long running (e.g. tickerplant, RDB, HDB etc.) or transient (e.g. a file loader may start at a certain time and run until completion)
- Where resilience is required, processes must be replicated
- Custom subscribers should be able to fail and recover. Recovery can be done from the tickerplant log or sometimes from the RDB.
- The RDB should be treated as a shared resource. It should not be relied upon to return the result of a query within a certain timeframe.
- It is usual to have multiple HDB processes as HDBs are usually lightweight

email:info@aquaq.co.uk

Big System Architecture

- Gateway processes are used to sit in front of HDB/RDBs and marshall queries across them. This allows the processes behind them to fail and restart transparently to clients
- Clients must use connection pools i.e. try process X, if not available try process Y etc.
- The whole system is usually replicated in different datacentres
- Sometimes queries can all be routed to one datacentre to allow maintainence to occur in the other

Maintainence and Housekeeping

- kdb+tick runs 24*7
- However, some housekeeping and maintainence tasks may be necessary to make sure it runs stably

1. Start-of-Day Checks
    - Ensure all processes are running
    - Ensure memory and CPU usage for processes are within acceptable bounds
    - Ensure enough disk space is available to capture a new day of data
    - Eusure the previous day of data has save down correctly
2. Tickerplant Log File Management
    - The full days worth of data is stored in the TP log file
    - Once the data is succesfully written to the HDB, these can be removed

email:info@aquaq.co.uk

Maintainence and Housekeeping

- However, it is usually better to compress and store them for a period

3. Log file management
    - Standard out/error will usually be redirected to log files
    - These should be compressed / deleted after a period of time
4. Periodic Bounce
    - It is usually a good idea to periodically bounce processes on a daily, weekly or monthly basis
    - Bounce will allow processes to clear out memory
    - Some processes may have subscriptions lists which need updated daily. This can usually be done by adding function calls, or bouncing the process

TorQ

email:info@aquaq.co.uk

TorQ

- AquaQ have developed and released (for free) TorQ - a kdb+ framework
- The aim is to bring together decades of knowledge in implementing production kdb+ systems, including:
  - process, code and configuration management
  - management of log info - errors, info, usage etc.
  - connection management and access controls
  - documentation and development tools
  - utilities for timer management, caching, async messaging, heartbeating etc.
  - process discovery, an advanced gateway, monitoring, tickerplant log replay, housekeeping
  - and much more...

TorQ

- Incorporates code from code.kx (some directly some modified)
- Extensively documented
- More can be found at: `https://github.com/AquaQAnalytics`

email:info@aquaq.co.uk

Useful Links and Further Reading

Useful Links

- http://www.aquaq.co.uk/q/
- http://www.listbox.com/subscribe/?listname=k4
- http://code.kx.com/wiki/Reference
- http://code.kx.com/wiki/Tutorials
- http://www.kdbfaq.com/
- http://lifeisalist.wordpress.com/
- https://groups.google.com/forum/#!forum/personal-kdbplus

email:info@aquaq.co.uk

Thank You For Listening!

For any further information or questions, please contact us:

e-mail: info@aquaq.co.uk
phone: +4402890511232