First Derivatives plc

Kx Technology Training Courses

q for gods

Exercises

Last Modified: 29 August 2008

Contents

IPC

Exercise A

1) Start one process on a port of your choice. Start another and open a connection to the server process.
2) Create a sample trade table on the server instance and insert a single row, sending the message as a string – just time, sym and price is fine.
3) Now insert multiple rows from the client using the second message format
4) Run a select query from the client and insert the results into a local variable.
5) From the client, create an empty quote table on the server – just a simple one with time, sym, bid & ask.
6) Define a simple function on the server and call it from the client.

Exercise B – Flushing

Uses the script IPC/ExKill.q

1) Start up one process. In another window use the script to kill the initial process you started.
2) What happens?
3) How can this be fixed? I.e you are still sending the asych kill. Clue: Now in 2.4 it can be fixed both synchronously and asynchronously.
4) What would be a better way to do this? Answer: an asynchronous kill.

Exercise C – Message Handlers

1) Modify the message handlers so that when a user opens or closes a connection, the information is stored in a table. The table should have the following columns: handle, time_of_connection, username, IP, time_of_close.

   Hints:
   .z.pc takes the message handler as its parameter
   "."sv string "i"$0x00 vs .z.a gives a more readable version of .z.a (IP address)

2) Modify .z.pg and .z.ps to create some simple logging so that all calls are inserted into an operations table. The operations table should contain the following columns: username, operation, duration, synch_or_asynch.

Attributes

1) Create a list of n distinct items. Measure the time to search for items at the front, middle and back of the list.

2) Repeat question 2 above, but applying the `s# attribute.

3) Repeat again, but with `u# applied.

4) Work out what length the list has to be for `u# attribute to be at least 1 microsecond faster than linear search (in the average case).

5) Repeat question 4, except compare `u# with `s#.

6) Show the speed up achieved when selecting from a table with a grouped column.

7) Show the memory overhead on a list with a `g# attribute.

8) Demonstrate, using a dictionary, the increase in speed achieved using the `u# attribute.

9) What would be the conditions required for a table to be a good candidate to have the `s# attribute applied to a column of a table?

10) Demonstrate using a table the difference made to search times by having the sort attribute.

    - What would be the conditions required for a table to be a good candidate to have the `u# attribute applied to a column of a table?
    - Prove to yourself that apply the `u# attribute to
    - What would be the conditions required for a table to be a good candidate to have the `g# attribute applied to a column of a table?
    - Demonstrate using a table the difference made to search times by having the grouped attribute.

Importing data

1. Import a text file. The file "trade.csv" is a comma-separated text file with the following fields:

   - date (list of dates)
   - sym (list of symbols)
   - size (list of integers)
   - price (list of floating-point values)
   - cond (list of characters)

   Import this file as a table into a q session, including all columns from the source file.

2. The folder "files" contains a series of csv files which contain trade data (the columns are the same as in exercise 1). Import each of the csv files and join the data into a single table. Sort the rows by date.

3. The file 'test.mdb' is a Microsoft Access file. Import all tables from this file into the q workspace.

4. Open an odbc connection to the file 'test.mdb'. List all the tables in the file, then retrieve all rows from the table 'Table1' where the value of the 'size' column is 1000, then close the connection.

Advanced web interface

1) Change .z.ph to log info of http requests

2) Change .z.ph so the below table displays properly in the web browser

```
tab:([]a:`$("<aa";"<table>";"&lt;";"<table\\>"))
```

   Hint: use ssr to change & to &, < to < and > to &rt; in .z.ph when the 1st char is "?"

3) Add refresh meta tag so results refresh every 10 seconds.

   Hint: change .h.html so it includes <meta http-equiv="refresh" content="10"> and update
. h.hp

4) Output tables as html striped tables.

   Hint: change .h.jx to use .h.htbl (from code.kx.com+Simon's example) defined before for tables, download plaid.js from http://www.clutch-inc.com/examples/plaid/ and include in the head of .h.html

q.k/q.q

1) Write a script which accepts 2 arguments: a text input file and an integer.

   The text file should contain one number (integer or float) per line (use Exercise1.txt).

   The script should read in the file in chunks to prevent memory overlaod, format the number in a string 10 characters wide with the specified number of decimal places, encode the resulting string using j10, and write the encoded string to standard out. It should then write the list of encoded strings (list of longs) to a binary file.

   The arguments are the input file and the number of decimal places.

   Startup command should look like this

   ```
   >q Exercise1.q -file Exercise1.txt -decimalPlaces 2
   ```

   Use the following functions from the .Q namespace: opt, j10, fmt, fs

   Note that .Q.j10 only encodes characters in the alphabet .Q.b6

2) Write a script which reads in the file created in Exercise 1, decodes the longs, casts the resulting strings to floats and rounds the floats down to integers.

   Output the square of each of the resulting integers. Optimise the calculation of the square roots, assuming there are duplicates in the integer list.

   Use the following functions from the .Q namespace : opt, x10, fu

   The single argument to the script is the input file

   ```
   q Exercise2.q -file Exercise1_output
   ```

Q Shortcuts (-x!)

1) Write an example usage of each q shortcut. Eg. for replaying a log file (-11!).

Each Prior Scan Over

Each Prior

1) Use each prior to get all the odd numbers up to 20.

2) Create a random list of 10 integers and find the indexes in that list where the next value is less than the preceeding value?

3) Given a simple table of, for example trade sizes for a single symbol, tbl:([]size:10?200 500 800 1000). Find the sizes that are double or greater than the prevailing size.

Scan

1) Create the list ("ab";"abc";"abcd";"abcde";"abcdef") using scan.

2) Use scan to fill any empty items in the list str:15?("";"abc";"mn";"wxyz") with the previous non-empty item.

Over

1) Find the total product of the list 2 4 8 16 32 64

2) Use scan or over to create a function that returns the first n Fibonacci numbers.

Joins

lj ij

1. Create a table, trade with columns: sym,price,size, of count 10. Sym column data should be generated from the following syms `A`B`C`D and suitable values should be chosen for price and size. Define the following table:
industry:([sym:`A`B`C`E];ind:`IT`Finance`Media`Transport)

2. Join the industry and trade tables to produce a table which is the intersection of the tables.

3. Join industry and trade so that an ind column is added to the trade table. All original entries that were in trade should still be present where there is no corresponding ind for a sym there should be a null value in the ind column. Call this table trade2.

4. Using lj join newInd:([sym:`A`C]ind:`Transport`Healthcare;Ex:`N`P`X) and trade2. Describe how the returned table differs from trade2.

5. Using ij join newInd and trade2. Describe how the returned table differs from trade2

6. Given:

```
tab0::([]a:1 2 3 2;b:`x`y`z`x)
tab1:([]a:3 3 2 2;c:`a`b`c`d)
```

   Produce:

```
a c b
-----
3 a z
3 b z
2 c y
2 c x
2 d y
2 d x
```

7. Below is table containing a record of changes in the telephone numbers of Tom and Bob during the last year.

```
phoneNum:([Name:`Tom`Bob`Tom`Bob;Date:2007.06.01 2007.06.01
2008.01.01 2008.06.01]phNum:336699 123456 999999 778899)
```

   Using phoneNum produce a table that shows Bob's telephone number on 2007.06.02, 2008.01.01, 2008.01.02 and today.

,' (join each)

1) Create few examples of using ,' (join each)

Pivoting

1) Consider the following table:

```
tab:([] c0:`a`b`c`a`b`c;c1:`b`a`a`c`c`b;c2:1 2 3 4 5 6) ;
```

   So that c0 and c1 only have the values `a, `b or `c, and c2 is numeric. How can this pivot table be constructed?

```
c0\c1 a    b    c
```

```
------------------------
a  0  1  4
b  2  0  5
c  3  6  0
```

i.e. c0 represents the rows and c1 the columns.

2) Now load in the script **transfers.q** and look at the table

```
q)transfers
Sell  Buy   Player  Amount
------------------------
ManU  Real  Ronaldo 75.5
Lvpl  ManU  Pennant 5.6
Real  Barca Guti    15.4
Barca Lvpl  Messi   34.89
Lvpl  Barca Leto    4.3
ManU  Real  Evra    7.2
```

Using the construction from the previous example, how could we construct a pivot table where the data values were the transfer amounts from the above table?

3) How could we include both of the transfers from ManU to Real in Exercise B?

Table Arithmetic

Populate table trade as below:

```
trade:([]time:`time$();sym:`symbol$();price:`float$();size:`int$())
syms:`IBM`GS`FD`KX
n:1000
insert[`trade;(n?"t"$.z.Z;n?syms;10*n?100.0;10*n?100)]
```

1) Multiply each price by 2.

2) Add 10.00 to the trade price where the symbol is FD.

3) Find the total running sum of shares traded from a particular basket of shares.

4) Find the running sym of shares traded by sym from a particular basket of shares.

5) Find the average price a bucket of symbols is trading at throughout the day, not taking volume into consideration.

6) Find the average price a symbol is trading at throughout the day, not taking the volume into consideration

7) Find the 3 step moving average of the price by sym, not taking the volume traded into consideration.

8) Find the average price traded throughout the day from a trade table, taking the volume of each trade into consideration.

9) Find the VWAP per sym throughout the day from a trade table.

Bucketing

1) Modify trade.q so that it inserts n rows into the table with the syms drawn from a sample set of 4 or 5. Making sure to have the resulting table ordered by time.

2) Using the table from 1 or otherwise. Write a query which takes the sum of total shares traded and places them in 5 minute buckets.

3) Do the same as above except in 1,2 and 10 minute buckets.

4) Using data from table in question 1 or otherwise. Write a query which takes the sum of total shares traded and place them in 10 second buckets.

5) Try the same as above in 60 second buckets. Is it the same as 1 minute buckets? When would one be used rather than the other?

6) Try the queries from questions 2->5 except have the buckets split by "x" time and by sym also.

7) Confirm that the sums of trades in buckets by sym match those not taking sym into consideration.

8) Construct the grid mentioned above, try to make it as general as possible, ideally as a function which takes the start and end times as well as the necessary granularity required to the query.

9) Using the queries from questions 2-7 and the grid from question 8, do the query mentioned in the note above question 8 for a query generating 3 minute buckets, 3 second buckets.

10) Fill in the nulls achieved in question 9 with zeros.

11) Using table

```
tbl:([]sym:20#`FD`KX`VC;time:09:00+til 20;price:20?100)
```

get the count between each of the following times:
09:00 09:02 09:05 09:10 09:17 09:20

12) Using tbl get the sum of the prices for the same time buckets as above.

13) Repeat question 2 this time include a column in the result that will show all the times included in the bucket.

14) Using tbl get the avg the prices for the same time buckets as question 1. This time show the end time of the bucket instead of the start time in the time column.

Correlation

Give examples of:

1. Corelated vectors -- cor 1
2. Negatively correlated vectors -- cor -1
3. Uncorrelated vectors -- cor ~ 0
4. Autocorrelation of random noise.
5. Recover signal from a sin wave with noise.
6. Price correlation across same sector average daily price over the past 50 days - e.g. Chevron(CVX)/Exxon(XOM)
7. Create a sector correlation matrix using iShare ETFs over 50 days

Functional/dynamic queries

Start a q session and load the script sampletables.q in order to create sample trade and quote tables for last 10 days. On loaded tables execute below tasks using functional form of queries.

1) Create in memory table t from quote data for one selected day and one symbol.

2) Delete from t table rows where bid>ask or any size is equal null

3) Remove column ex from table t

4) Now for modfied t table create additional column (ask%bid) for rows where ask is more than 10% grater than bid

5) Return smallest and largest price from trade table for last 3 days

6) Select all IBM trades with a price less than or equal to 1.5

7) Select the last price for each date, sym in hourly buckets

8) Calculate the hourly mean, variance and standard deviation of the price for AIG

9) Daily Spread (average bid-ask) for CSCO for 5 last days

Load the q script required_tables.q. This script defines 2 tables: quote and execution. On loaded date execute following tasks.

10) Select the maximum ask price by sym from the quote table.

11) Select the minimum bid price by sym from the quote table.

12) Select the maximum ask price by sym and exchange from the quote table.

13) Select the last bid price and the last ask price by exchange from the quote table.

14) Select the first ask price and the minimum bid price by sym and exchange from the quote table. The resultant columns should be named first_ask and min_bid respectively.

15) Select the last time by sym and account from the execution table. The resultant column should be called last – time.

16) Select the time weighted average price by sym from the execution table. The resultant column should be called wavg – price.

17) Select the maximum price and the minimum price by strategy id and inserted_by from the executions table. The resultant columns should be named max_price and min_price respectively.

18) Some of the executions were executed at a price greater than 1.2 and some were executed at a price below 1.2. Create a summary table with 2 columns as follows. The 1st column should represent this threshold value and be called thresh and the 2nd column should be the associated execution_ids.

19) Execute the maximum price and the last execution_id by sym and clinet_id from the execution table. The 2 resultant columns should be labelled max_price and last_exec respectively.

Advanced Language Interfaces

C API

1) Which files are required to interface q with C

2) What type of applications can use the C API

3) How is the execution flow through the C API?

4) Which steps are required to communicate with a q process (TP) using the C API?

5) Is there an asynchronous or a synchronous connection with a q process?

6) Which errors are reported on asynchronous queries?

7) How are the errors reported on synchronous queries?

8) If it's possible to use in a q process, functions defined in C, how can we do that?

9) In which way we can map a function "sum" with 3 parameters defined in "path/shared_lib" C shared library?

10) May we call from the C functions defined in a shared library, another functions defined in the q process referring it? How can we do that?

11) How many arguments we may pass to a q function?

12) How is represented a K object in C?

13) Which is the correspondence between q types (K objects) and their value representation in C:

14) Describe some useful functions from C API:

15) Create a stand-alone application which connect to a q process listening to port 1234 on "localhost".

    The application must print "Hello_world" message to the q console and exit if no problems occurs.

    Start a q process on the specified port, compile and run the application.

Importing Foreign Libraries

1) Import a simple dll into q. Create a call back function in q from c.

Tables on Disk

1) Add attribute `p (partitioned) to one column on disk.

   n.b. Now use script exDb.q to create example database consisting of two partitioned tables (trade and quote) and one static table (productInfo) used in below exercises:

2) Change name of static table (productInfo -> symInfo)

3) Create symbol list of date partitions.

4) Change name of partitioned table (quote -> marketData)

5) Backfill trade table to all partitions

6) Get list of column names from schema (.d file) for trade

7) Change order of columns (marketData: reverse column order

8) Change name of column (trade: price -> val)

9) Make sym column of trade upper case

10) Remove size column from trade

11) Create a function to list contents of directory and indicate whether each is a directory or file

Logging & Checkpointing

1) Create .qdb and .log, delete .qdb as below:

```
$ q checkpoint -l
q)t:([]x:`A`B`C`D;y:1 2 3 4)
q)\l
q)0"insert[`t;(`E;5)]"
,4
q)t
x y
---
A 1
B 2
C 3
D 4
E 5
q)\\
$ rm checkpoint.qdb
$ q checkpoint –l
'type
```

   Recover all updates to the database.

2) Look at the .log file – write an alternative method of writing to the .log file.

3) Look at the .log file – write an alternative method of clearing the .log file.

4) Look at the .qdb file – write an alternative method of writing the .qdb checkpoint file.

5) Look at the .qdb file – write an alternative method of clearing the .qdb checkpoint file.

6) Process is subscribed to tickerplant. Set up end of day process – to checkpoint and clear log file:

   a. Using standard command.
   b. Using methods written in 2-5. Each day should have its own checkpoint file e.g. checkpoint2008.08.06.qdb, checkpoint2008.08.07.qdb.
   c. Write functions to reload .qdb and replay .log files.

7) Database contains keyed and unkeyed tables – only checkpoint keyed tables.