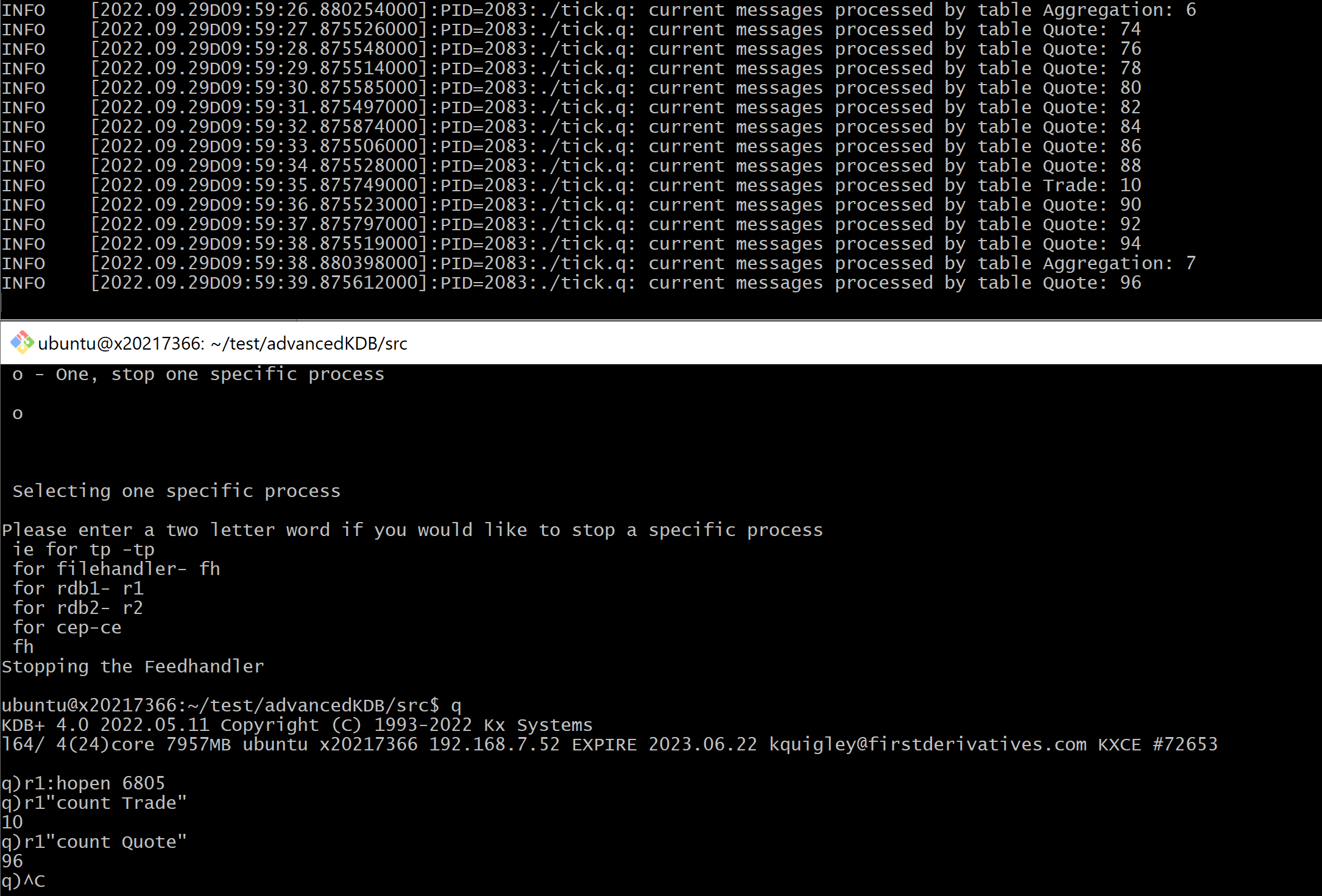
*Advanced KDB Solutions – Kevin Quigley*

*Hello and welcome to my repo generating solutions to the Advanced KDB exam from First Derivatives*

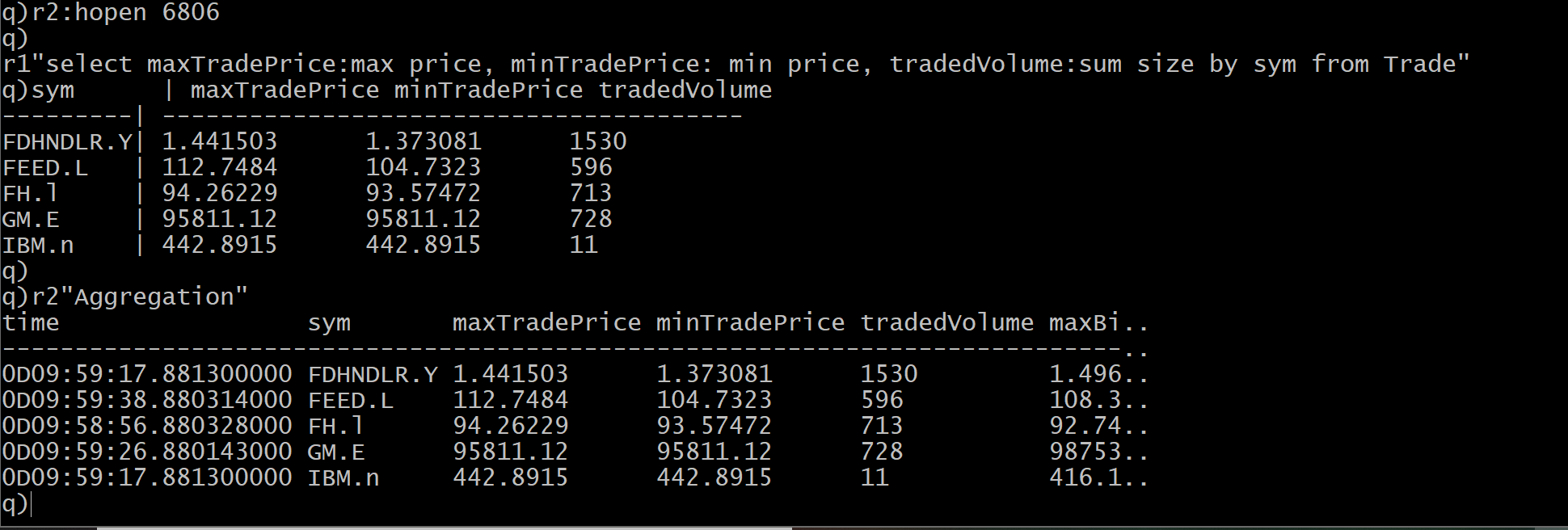
*Q Common libraries taken from* [*https://github.com/BuaBook/kdb-common/tree/master/src/cron.q*](https://github.com/BuaBook/kdb-common/tree/master/src/cron.q)

*Q Tick libraries taken from* [*https://github.com/KxSystems/kdb-tick*](https://github.com/KxSystems/kdb-tick)

The logging has been updated to refect the number of lines added to each table:-



There was an issue with the CEP pushing an update if there was a single line of data, which has now been fixed and the aggregation tables match:-



Initialization

1. Unzipping. The unzip location for the Advanced kdb folder can be set anywhere. *For this example, the project is unzipped into the home-directory (~).* After unzipping, cd to *~/advancedKDB-master/*. Most process files are located in the *~/advancedKDB-master/src* folder.
2. cd to the src file in *~/advancedKDB-master/src.*
3. Ports are set in the *~/advancedKDB-master/src/env.sh* file, and can be changed there
4. The system can be started with bash START.sh in the src folder. Similarly, it can be tested using bash TEST.sh, and stopped using bash STOP.sh

***Q1 –Tickerplant***

The tickerplant has had it’s tick updated to only record every time it recieves an update to a table

Tickerplant is defined in *~/advancedKDB-master/src/tick.q*. The schema for all other tables is defined in sym.q. During startup, the ports are set via env.sh. The port values are set in /env.sh

To launch the tickerplant without a bash script, execute this line in the src folder mentioned above:-

q ./tick.q "sym" $(pwd)/../raw -p 6800 -t 100

***Q2 RDB***

The RDBs are defined in *~/advancedKDB-master/src/rdb.q* and *~/advancedKDB-master* src/rdb2.q

When the RDB is restarted, it reloads the historical data. The total number of Trades processes in the table may outnumber the number of ones processed by the logs if the table was reloaded, as the rdb reloads the Trade and Quote table from the rdb.

The Aggregation table is not saved down, and so should only keep a record of the aggregated trades that were most recently processesed by the CEP.

***Q3 Feed handler***

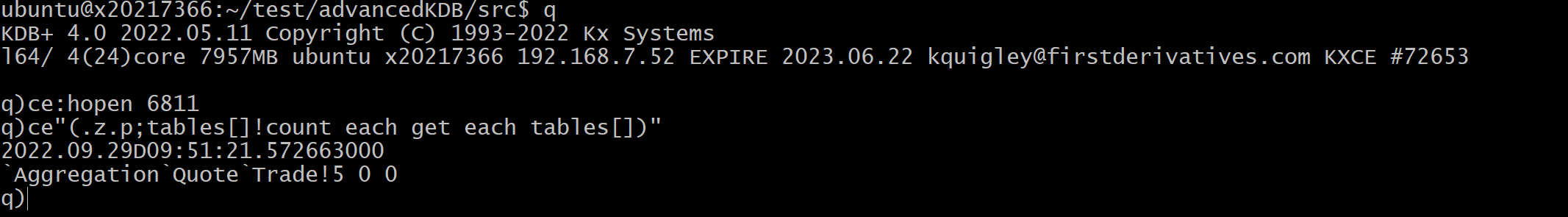
Feedhandler is defined in *~/advancedKDB-master/src/fh.q*

***Q 4 CEP***

*CEP is defined in ~/advancedKDB-master/src/cep.q*

*For TP logging every minute .logs/data/tp.log*

Now when the CEP receieves an Quote or Trade, the relevant data is taken from the tables. When the data is then pushed to the TP, it is deleted from memory in the CEP.



***Q5 Logging***

Location:*- ~/advancedKDB-master/log4q-master/log4q.q*

Logging for the instances in my project use log4q. An example of a coded error message which would work is as follows, after connecting to an instance:-

*.log4q.INFO("TEST")*

Will add the following to the log of the associated instance

*INFO [2022.09.26D20:03:09.303421000]:PID=41010:./fh.q: TEST*

*KDB+ 4.0 2022.05.11 Copyright (C) 1993-2022 Kx Systems*

*l64/ 4(24)core 7957MB ubuntu x20217366 192.168.7.52 EXPIRE 2023.06.22 kquigley@firstderivatives.com KXCE #72653*

***Q6 Startup Shutdown scripts***

*Start, stop and test scripts are divided into three separate scripts*

*Upon starting a process, the pid for the process will be stored in /adv\_kdb/logs/pids*

*Run commands:*

*.cd ~/advancedKDB-master/src/*

*Bash START.sh -> input y to start all process, input n with any combination of two letter prompt to start a different process, ie tp starts the tickerplant. The instructions are mentioned in the prompts*

*Bash STOP.sh -> input y to stop all process, input n with any combination of two letter prompt to stop a different process, ie tp starts the tickerplant. The instructions are mentioned in the prompts*

*Bash TEST.sh -> input n with any combination of two letter prompt to test a different process, ie tp starts the tickerplant. The instructions are mentioned in the prompts*

Normally

***Q7 TP Replay***

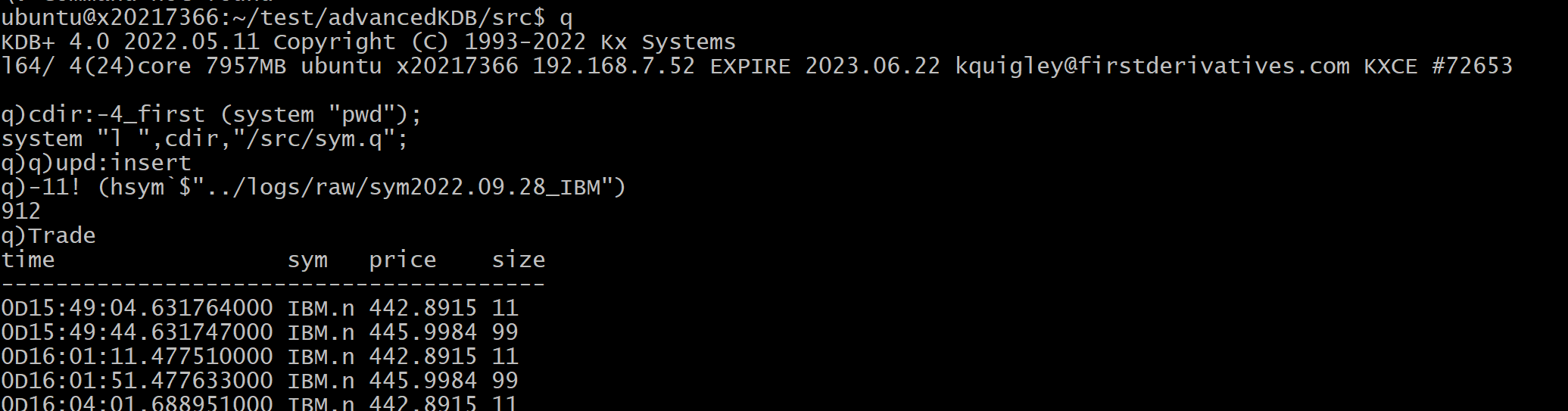
*Location ~/advancedKDB-master/src/tpLogReplay.q*

*Output Location ~/advancedKDB-master/logs/raw/symYYYY.MM.DD\_IBM*

Example, cd to *~/advancedKDB-master/src/*

Execute *q ./tpLogReplay.q sym2022.09.28*

To replay the created log, follow as shown below



***Q8 CSV Loader***

Location: *./src/ csvFileLoad.q*

Run Command: *q csvFileLoad.q*

Output Location – The Quote table on the RDB

TP must be started before running the loader and the table is pusblished to the rdb following this. To view the table, execute

*q*

*h:hopen 6805*

*h”Quote”*

***Q9 HDB Compression***

*Location: ./src/compress\_hdb.q*

*Run command: hdbCompress.q 2022.09.27*

*Output Location: ./hdb/*

***Q10 Effect of schema change***

1. The symfile (src/sym.q) would have to updated
2. Changing the names of the columns could result in issues in compression, if the date or symbol column is changed.
3. Re-ordering the columns could cause issues with functions dependend on specific columns
4. the old hdb would need to backfilled with the new order/ new column. This will involve modifying the file on each partition to account for the new column
5. The TP would need to be taken offline, after EOD. This would be the best time to execute such a change, as opposed to during a release.
6. If the table is changed during the day, all subscriber which hold data will need to be updated in memory.

***PART 2:- Debugging***

***Q1. TP\_log***

Running the line ([]logResult: (get `:tplog) ; columnTypes: ( { type each first ( get `:tplog ) [ x ] [ 2 ] } each til 10))

Yields the following issues:-

* The log entry is missing rows between 6-8
* The type of the size column is inconsistant between rows
* Some of the types in the sym columns are inconsistant

Due to the very small size of the log file, my solution is to read the old broken log into memory, modify fields where required and the save down a new log file ‘new\_log’. This can be renamed to tplog if required.

**Steps:**

* Assign the old log table

oldLog: get `:tplog

* Correcct the corrupted line which was split across multiple lines

newLog:oldLog[0+til 6],(enlist fixedRecord:oldLog[6],oldLog[7],oldLog[8]), enlist oldLog[9]

* Update datatypes on sym column

newLog: {(`upd;`trade;@[x;exec c from meta x where t ="c",c=`sym; {`$x } each ]) } each last each newLog

* Ensure that the types of the numerical column are the cast to float

newLog:{(`upd;`trade;@[ x; exec c from meta x where t = "f",c=`size;{"j"$x} each])} each last each newLog

* Resave the table

`:newLog set ()

h:hopen `:newLog

h newLog

Trade:([] sym:`$();price:"f"$();size:"j"$())

upd:insert

-11!`:newLog

Following above the trade table can be queried. The steps above are saved a q file, logDebugger.q.

**Part 2 – located in db folder- tableDebugger.q - Splay Table problem**

//Moving the .d file in table number 1

system"cp ./t2/.d ./t1/.d"

//Fixing the price columns in table 2

priceCol:get hsym `$"./t2/price

priceCol:(priceCol,0.0)

(hsym `$"./t2/price") set priceCol

//Fixing table 3 enumeration

\l t3

.Q.en[`:t3;t3]

**Debugging Part 3:- Blocking calls**

If multiple users are connecting to a single kdb process each query may be ran an inefficient manner, or it could be the case that slowness is being caused by having each query executed one after another

The most obvious solution is to have two or more hdb’s located in the same directory. Having a single gateway process which routes traffic and balances it between each hdb would help to minimize slowness.

Another advantage to routing queries through a gateway is that by setting the api calls, you can make sure that there are limits being applied to the calls, ie no query that calls to a splayed table without having the date as the first paramater, or setting date limits on queries on very large tables.

**Debugging Part 4:- Query Performance**

Apologies for missing this. First time I missed it, the second time I think the push to the github repo went to a different branch. If you review my videos you see it’s working correctly, but below is the push I meant to make.

StringtoDate:{@[x;where (type each x) in 10h;"D"$]} raze 500000#enlist("2010.01.01";2010.01.02)

***API***

**API Part 1**

Completed using python3

Run cmd: **python2 csvPublisher.py** located in /advancedKDB/src/

TP is on port 6800.

This publishes the table Quote.csv, which is located in the same folder, to the TP

**API Part 2**

The C-API is located in the advancedKDB/src/CAPI folder, it needs to be build with the locally build c-libaray for kdb (k.h, will error unless this is build correctly).

File is built using

gcc -o publish publish.c c.o -lpthread

Executed using the following in the CAPI folder

./publish

**API Part 3**

The html is located in the folder /advancedKDB/src/html/.

First the q query process is launched using

q queryTrades.q

On a remote machine, the html service can be run with the line:-

python3 -m http.server 1234

To query from a remote machine, enitialize port forwarding:-

ssh -i .sshkeys/id\_rsa\_server -N -L localhost:8787:localhost:1234 ubuntu@87.44.4.70

ssh -i .sshkeys/id\_rsa\_server -N -L localhost:6700:localhost:6700 ubuntu@87.44.4.70

On your local machine, open localhost:8787 in browser and kdb commands can be run. Note that the permissions between the python process and the html process must be be set to allow traffic

<http://localhost:8787/>