AquaQ TorQ: Finance Starter Pack

email:
support@aquaq.co.uk

web:
www.aquaq.co.uk

Revision History

Revision	Date	Author(s)	Description
1.0	2014.12.16	Kevin Piar and Jonny Press	First version released

Contents

Chapter 1

Company Overview

AquaQ Analytics Limited is a provider of specialist data management, data analytics and data mining services. We also provide strategic advice, training and consulting services in the area of market-data collection to clients predominantly within the capital markets sector. Our domain knowledge, combined with advanced analytical techniques and expertise in best-of-breed technologies, helps our clients get the most out of their data.

The company is currently focussed on four key areas, all of which are conducted either on client site or near-shore:

- Kdb+ Consulting Services: Development, Training and Support. We are an official implementation and training partner of Kx Systems;
- Real Time GUI Development Services;
- SAS Analytics Services;
- Providing IT consultants to investment banks with Java, .NET and Oracle experience.

The company currently has a headcount of 30 consisting of both full time employees and contractors and is actively hiring additional resources. Some of these resources are based full-time on client site while others are involved in remote/near-shore development and support work from our Belfast headquarters. To date we have MSAs in place with 6 major institutions across the UK and the US.

Please feel free to contact us if you feel we may be able to assist you with your kdb+, data or analytics needs.

info@aquaq.co.uk

Chapter 2

TorQ Demo Pack

The purpose of the TorQ Demo Pack is to set up an example TorQ installation and to show how applications can be built and deployed on top of the TorQ framework. The example installation contains all the key features of a production data capture installation, including persistence and resilience. The demo pack includes:

- a dummy data feed
- a resilient kdb+ stack to persist data to disk and to allow querying across real time data and historic data
- basic monitoring with notifications via email
- automated report generation

Once started, the system will generate dummy data and push it into a database. The system will operate 24*7 and will tidy up old files as it goes.

Further information about each feature can be found in the TorQ Manual.

Chapter 3

Getting Started

3.1 Requirements

The TorQ Demo Pack will run on Windows, Linux or OSX. It contains a small initial database of 130MB. As the system runs, data is fed in and written out to disk. We recommend that it is installed with at least 2GB of free disk space, on a system with at least 4GB of RAM. Chrome and Firefox are the supported web browsers.

It is assumed that most users will be running with the free 32-bit version of kdb+. TorQ and the TorQ demo pack will run in exactly the same way on both the 32-bit and 64-bit versions of kdb+.

3.2 Installation and Configuration

3.2.1 Installation

1. Download and install kdb+ from Kx Systems[1]
2. Download the main TorQ codebase from here[2]
3. Download the TorQ Demo Pack from *** insert link ****
4. Unzip the TorQ package
5. Unzip the Demo Pack over the top of the main TorQ package

[1] http://kx.com
[2] https://github.com/AquaQAnalytics/TorQ/archive/master.zip

3.2.2 Configuration

1. Modify config/process.csv to specify the host name of the local machine. In the "host" column of the csv file, input the hostname or IP address of the machine where all the processes will run. On Linux or OSX this can also be done by running the hostname.sh script from the terminal prompt.

If you want to generate emails from the system, you will additionally have to:

1. Modify DEMOEMAILRECEIVER environment variable at the top of start_torq_demo.sh, start_torq_demo_osx.sh or start_torq_demo.bat
2. Add the email server details in config/settings/default.q. You will need to specify the email server URL, username and password. An example is:

```
// configuration for default mail server
\d .email
url:`$"smtp://smtp.email.net:80"        // url of email server
user:`$"testaccount@aquaq.co.uk"        // user account to use to send emails
password:`$"testkdb"                    // password for user account
```

3.3 Start Up

3.3.1 Windows

Windows users should use start_torq_demo.bat to start the system, and stop_torq_demo.bat to stop it. start_torq_demo.bat will produce a series of command prompt. Each one of these is a TorQ process. *** SEE IMAGE BELOW ****

3.3.2 Linux and OSX

Linux users should use start_torq_demo.sh to start the system, and stop_torq_demo.sh to stop it. OSX users should use start_torq_demo_osx.sh to start the system, and stop_torq_demo.sh to stop it. The only difference between the respective start scripts is how the library path environment variable is set. The processes will start in the background but can be seen using a ps command, such as

```
aquaq> ps -ef | grep 'torq\|tickerplant'
aquaq    4810 16777  0 15:56 pts/34   00:00:00 grep torq\|tickerplant
aquaq   25465     1  0 13:05 pts/34   00:00:05 q torq.q -load code/processes/
    discovery.q -p 31000 -U config/passwords/accesslist.txt -o 0
aquaq   25466     1  0 13:05 pts/34   00:00:29 q tickerplant.q database hdb -p 31100
    -U config/passwords/accesslist.txt -o 0
aquaq   25478     1  0 13:05 pts/34   00:00:17 q torq.q -load code/processes/rdb.q -p
     31200 -U config/passwords/accesslist.txt -o 0 -g 1 -T 30
aquaq   25479     1  0 13:05 pts/34   00:00:04 q torq.q -load hdb/database -p 31300 -
    U config/passwords/accesslist.txt -o 0 -g 1 -T 60 -w 4000
aquaq   25480     1  0 13:05 pts/34   00:00:05 q torq.q -load hdb/database -p 31301 -
    U config/passwords/accesslist.txt -o 0 -g 1 -T 60 -w 4000
```



```
aquaq   25481     1  0 13:05 pts/34   00:00:06 q torq.q -load code/processes/gateway.
    q -p 30000 -U config/passwords/accesslist.txt -.servers.CONNECTIONS hdb rdb -o 0
    -g 1 -w 4000
aquaq   25482     1  0 13:05 pts/34   00:00:06 q torq.q -load code/processes/monitor.
    q -p 30200 -U config/passwords/accesslist.txt -o 0
aquaq   25483     1  0 13:05 pts/34   00:00:07 q torq.q -load code/processes/reporter
    .q -p 30500 -U config/passwords/accesslist.txt -o 0
aquaq   25484     1  0 13:05 pts/34   00:00:04 q torq.q -load code/processes/
    housekeeping.q -p 30400 -U config/passwords/accesslist.txt -o 0
aquaq   25485     1  0 13:05 pts/34   00:00:05 q torq.q -load code/processes/wdb.q -p
     31450 -U config/passwords/accesslist.txt -o 0 -g 1
aquaq   25486     1  0 13:05 pts/34   00:00:13 q torq.q -load code/processes/wdb.q -p
     31400 -U config/passwords/accesslist.txt -o 0 -g 1
aquaq   25547     1  0 13:05 pts/34   00:00:13 q torq.q -load tick/feed.q -p 30700 -o
     0
```

3.3.3 Check If the System Is Running

TorQ includes a basic monitoring application with a web interface, served up directly from the q process. The monitor checks if each process is heartbeating, and will display error messages which are published to it by the other processes. New errors are highlighted, along with processes which have stopped heartbeating. Data is published to the front end, rather than polled from the front end.

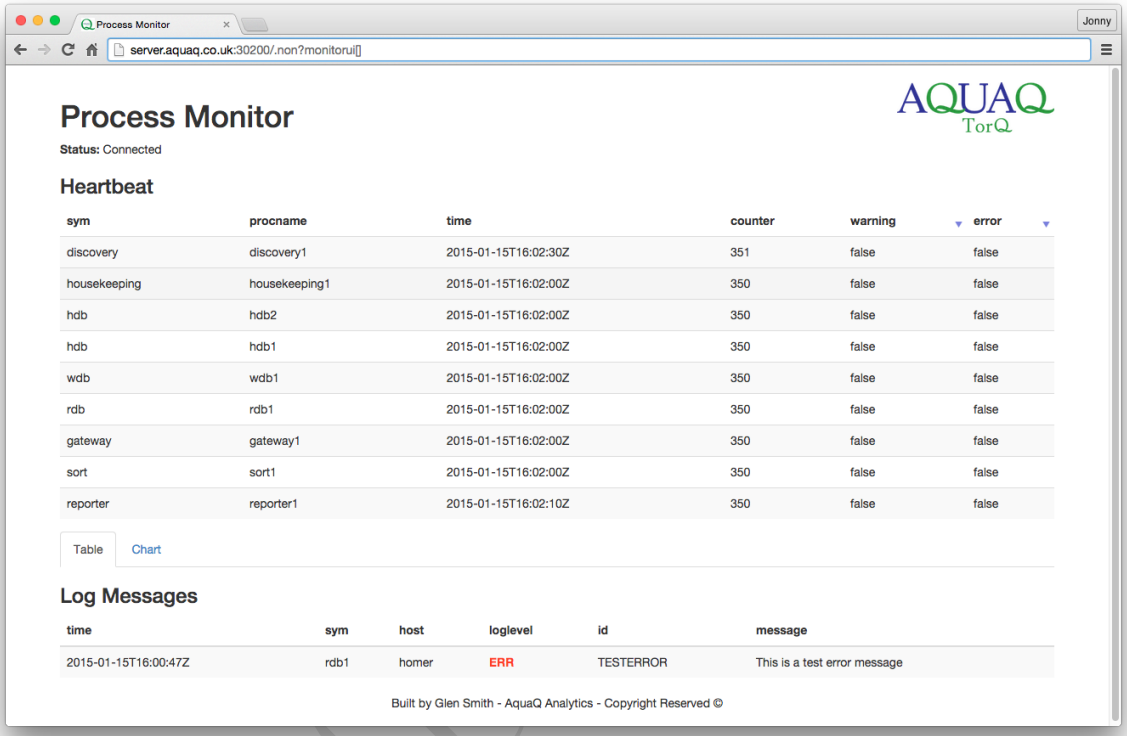

Figure 3.1: Monitor UI

The monitor UI can be accessed at the address http://hostname:30200/.non?monitorui[] where hostname is the hostname or IP address of the server running the monitor process. Note that the hostname resolution for the websocket connection doesn't always happen correctly- sometimes it is the IP address and sometimes the hostname, so please try both. To see exactly what it is being returned as, open a new q session on the same machine and run:

```
q)ss[html;"KDBCONNECT"] _ html:`::30200:admin:admin "monitorui[]"
"KDBCONNECT.init(\"server.aquaq.co.uk\",30200);\n</script>\n    </body>\n</html>\n"
```

3.3.4 Connecting To A Running Process

Any of the following can be used to easily interrogate a running q process.

- another q process, by opening a connection and sending commands
- qcon[3]

[3]can be downloaded from the OS specific directory in `http://code.kx.com/wsvn/code/kx/kdb+/`

- an IDE[4]

The remainder of this document will use either qcon or an IDE. Each process is password protected but the user:password combination of admin:admin will allow access.

3.3.5 Testing Emails

If you have set up emailing, you can test is using the .email.test function. This takes a single parameter of the email address to send a test email to. It returns the size of the email sent in bytes upon success, or -1 for failure.

```
aquaq$ qcon :31200:admin:admin
:31200>.email.test[`$"testemail@gmail.com"]
16831i
```

To extract more information from the email sending process, set .email.debug to 2i.

```
:31200>.email.debug:2i
:31200>.email.test[`$"testemail@gmail.com"]
16831i
```

3.4 Trouble Shooting

The system starts processes on ports in the range 30000 to 32000. If there are processes already running on these ports there will be a port clash- change the port used in both the start script and in the process.csv file.

All the processes logs to the $KDBLOG directory. In general each process writes three logs: a standard out log, a standard error log and a usage log (the queries which have been run against the pro cess remotely). Check these log files for errors.

3.4.1 Debugging

The easiest way to debug a process is to run it in the foreground. By default, TorQ will redirect standard out and standard error to log files on disk. To debug a process, start it on the command line (either the command prompt on Windows, or a terminal session on Linux or OSX) using the start up line from the appropriate launch script. Supply the -debug command line parameter to stop it redirecting output to log files on disk.

If the process hits an error on startup it will exit. To avoid this, use either -stop or -trap command line flag. -stop will cause the process to stop at the error, -trap will cause it to trap it and continue loading. An example is below.

[4] http://code.kx.com/wiki/Startingkdbplus/introduction


```
q torq.q -load code/processes/rdb.q -p 31200 -U config/passwords/accesslist.txt -o 0
    -debug -stop
```

3.5 File Structure

The file structure can be seen below.

```
|-- README_demo.txt
|-- code
|   |-- common
|   |   `-- u.q                 <- kdb+ tick pubsub script
|   |-- hdb                     <- extra functions loaded by hdb procs
|   |   `-- examplequeries.q
|   `-- rdb                     <- extra functions loaded by rdb procs
|       `-- examplequeries.q
|-- config
|   |-- application.txt         <- TorQ demo pack banner
|   |-- compressionconfig.csv   <- modified compression config
|   |-- housekeeping.csv        <- modified housekeeeing config
|   |-- passwords
|   |   |-- accesslist.txt      <- list of user:pass who can connect to proccesses
|   |   `-- feed.txt            <- password file used by feed for connections
|   |-- process.csv             <- definition of type/name of each process
|   |-- reporter.csv            <- modified config for reporter
|   `-- settings                <- modified settings for each process
|       |-- compression.q
|       |-- feed.q
|       |-- gateway.q
|       |-- monitor.q
|       |-- rdb.q
|       |-- sort.q
|       `-- wdb.q
|-- hdb                         <- example hdb data
|   `-- database
|       |-- 2015.01.07
|       |-- 2015.01.08
|       `-- sym
|-- hostname.sh                 <- script to reset hostname
|-- start_torq_demo.bat         <- start and stop scripts
|-- start_torq_demo.sh
|-- stop_torq_demo.bat
|-- stop_torq_demo.sh
|-- tick                        <- kdb+ tick
|   |-- database.q              <- schema definition file
|   |-- feed.q                  <- dummy feed from code.kx
|   |-- r.q
|   `-- u.q
`-- tick.q                      <- kdb+ tick
```

The Demo Pack consists of:

- a slightly modified version of kdb+tick from Kx Systems
- an example set of historic data
- configuration changes for base TorQ

- additional queries to run on the RDB and HDB
- start and stop scripts

3.6 Make It Your Own

The system is production ready. To customize it for a specific data set, modify the schema file and replace the feed process with a feed of data from a live system.

Chapter 4

Architecture

This section will be used to describe the different features included within the demo pack. For exact implementation details please refer to the TorQ manual, review the code or contact info@aquaq.co.uk.

4.1 Processes

The architecture of the demo system is as below.

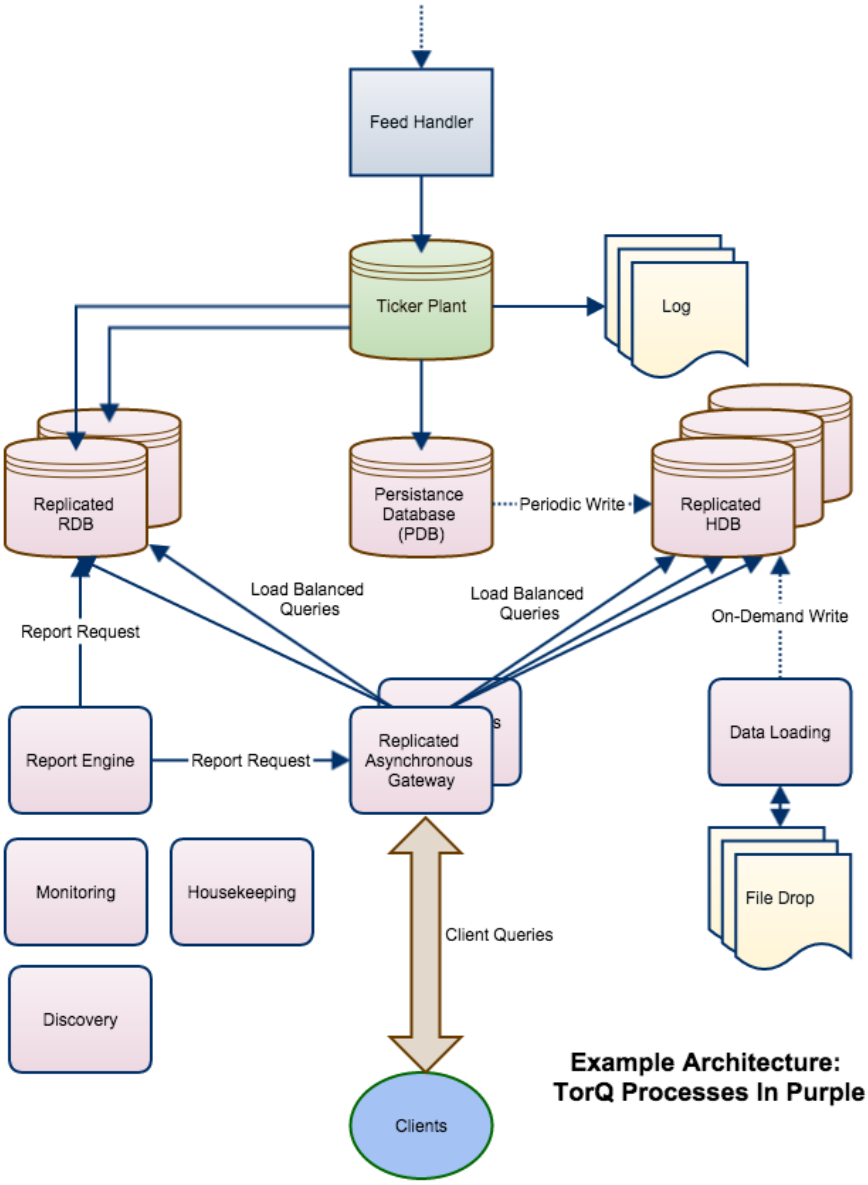

Figure 4.1: Demo Data Capture

4.1.1 Feed

The feed comprises two randomly generated tables, trade and quote. These tables have 'ticks' generated by feed.q which are pushed to the tickerplant in batches every 200 milliseconds. A large batch is pushed initially then smaller batches after it. The timestamps are local time.

The schema definitions can be seen below:

```
quote:([]time:`timestamp$(); sym:`g#`symbol$(); bid:`float$(); ask:`float$(); bsize:`
    long$(); asize:`long$(); mode:`char$(); ex:`char$())
trade:([]time:`timestamp$(); sym:`g#`symbol$(); price:`float$(); size:`int$(); stop:`
    boolean$(); cond:`char$(); ex:`char$())

meta quote
c    | t f a
-----| -----
time | p
sym  | s   g
bid  | f
ask  | f
bsize| i
asize| i
mode | c
ex   | c

meta trade
c    | t f a
-----| -----
time | p
sym  | s   g
price| f
size | i
stop | b
cond | c
ex   | c
```

4.1.2 Tickerplant

The tickerplant is the standard kdb+tick tickerplant, with a modification to apply timestamps as timestamp type rather than timespan. The tickerplant log file will be written to hdb/database.

4.1.3 RDB

The RDB is a TorQ process which holds data for the current GMT day in memory. Unlike kdb+tick, it does not persist data to disk at end-of-day.

4.1.4 WDB and Sort Processes

The WDB is a specilized database process which subscribes to a tickerplant and periodically persists data to disk. At EOD this data is used to create the HDB partition. It has been configured to operate in conjuction with a sorting process which sorts the data it writes to disk.

The sorting process is configured to sort by sym(`p#) and time, although this can be configured on a per-table basis in $KDBCONFIG/sort.csv

4.1.5 Gateway

The gateway connects to the RDB and HDB processes and runs queries against them. It can access a single process, or join data across multiple processes. It also does load balancing and implements a level of resilience by hiding back end process failure from clients.

4.1.6 Report Engine

The Report Engine runs queries on a schedule against specific back end processes, including gateways. Once the report is complete the result can be further processed, with available actions such as emailing it out or writing it to a file. This has been used to implement some basic monitoring checks and run some end of day reports. The configuration file is in $KDBCONFIG/reporter.csv.

4.1.7 Housekeeping

The housekeeping process is used to maintain some of the files written to disk by TorQ. In the demo we use it to archive tplogs and both archive and eventually remove log files from the TorQ working directories.

The process has been configured like so:

```
zip,{KDBLOG}/,*.log,,10
rm,{KDBLOG}/,*.gz,,30
zip,{KDBHDB}/,database20*,,1
```

The first line can be translated to mean 'Compress the files in the KDBLOG/ path, matching the *.log pattern, excluding no files and where the files are older than 10 days'

Combined with the other lines, the system will compress process logs after 10 days, delete compressed process logs after 30 days and compress tplogs after 1 day.

The compression process will check for work to be done everyday at 1200 local time.

4.1.8 Compression

The compression process is used to periodically scan the hdb directory for columnar binary files to compress. The compression settings are defined in $KDBCONFIG/compressionconfig.csv. This allows configuration of compression parameters on a per table, column and age basis.

It is intended to be used with a scheduling program like cron. By default it is a transient process as it will start up, check for files to compress, does any work required and then dies.

4.1.9 Discovery

The discovery process is used by the other processes to locate other processes of interest, and register their own capabilities.

4.1.10 Monitor

The monitor process is a basic monitoring process to show process avaiability via heartbeating, and to display error messages published by other processes.

4.2 What Advantages Does This Give Me?

A standard kdb+tick set up is great for a lot of installations but some customers have modified it substantially to fit their needs.

4.2.1 End Of Day

In a standard kdb+tick setup, the end-of-day event is time consuming and the data is unavailable as it is written from the RDB memory to the HDB disk. With the above setup, this outage is minimized in the following ways:

- Faster end-of-day as data is written periodically to disk throughout the day
- No back-pressure (slow subscriber problem) on the tickerplant as the RDB doesn't write to disk, and the WDB doesn't do the time consuming sort on the data
- "Yesterday's" data is available in the RDB until the end-of-day operation is complete, meaning no data outage

4.2.2 Gateway: Resilience, Load Balancing and Parallel Access

kdb+tick doesn't include a gateway as standard. There are some examples on code.kx, but production gateways are generally non trivial to write. The TorQ gateway ensures

- Backend processes can be replicated as required. If one process fails, another will take over transparently to the client

- New processes can be started intraday and will be automatically available via Discovery Service notifications
- Queries are run in parallel and load balanced across back end processes- multiple clients can query at once

4.2.3 Supportability

TorQ adds a layer of standard tools to aid system supportability on top of kdb+tick.

- Common directories for loading code in a fault tolerant way
- Output and error log messages are timestamped and standardized
- Log messages can be published to external applications
- All client queries are logged and timed, and can be externally published
- Monitoring checks are incorporated
- Email notifications are incorporated
- Housekeeping automatically executed

Chapter 5

Have a Play

5.1 Gateway

5.1.1 Queries

Some example queries have been implemented on the RDB an HDB processes. These are defined in $KDBCODE/rdb/examplequeries.q and $KDBCODE/rdb/examplequeries.q. These can be run directly on the processes themselves, or from the gateway which will join the results if querying across processes. To test, connect to the gateway process running on port 30000 from q process, qcon or from an IDE. An example is shown below running from an IDE.

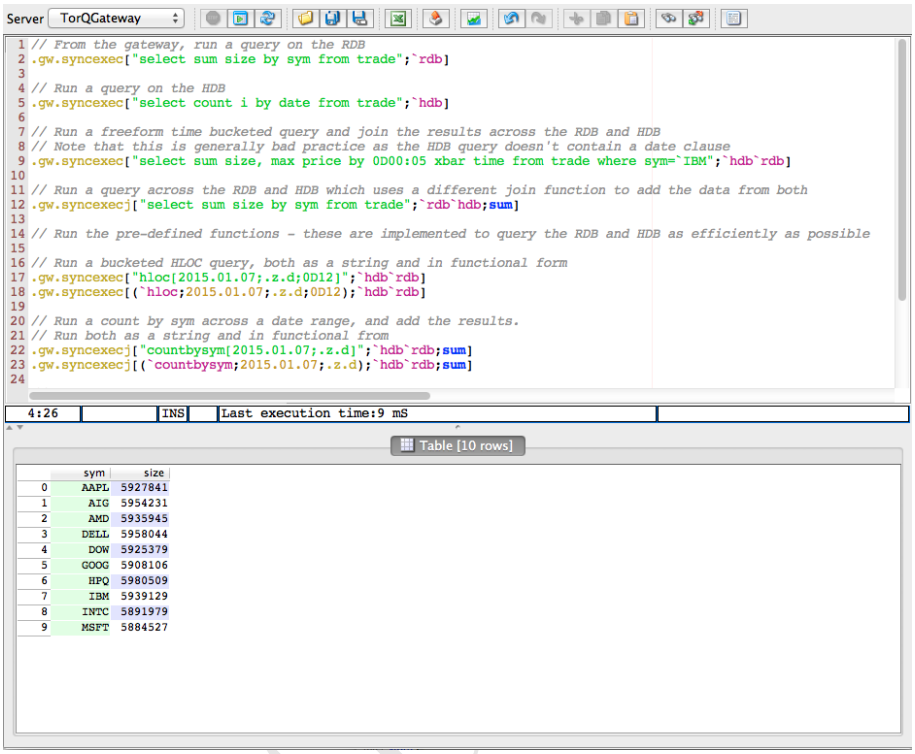

Figure 5.1: Gateway Queries Running from an IDE

Example queries are listed below.

```
// From the gateway, run a query on the RDB
.gw.syncexec["select sum size by sym from trade";`rdb]

// Run a query on the HDB
.gw.syncexec["select count i by date from trade";`hdb]

// Run a freeform time bucketed query and join the results across the RDB and HDB
// Note that this is generally bad practice as the HDB query doesn't contain a date
    clause
.gw.syncexec["select sum size, max price by 0D00:05 xbar time from trade where sym=`
    IBM";`hdb`rdb]

// Run a query across the RDB and HDB which uses a different join function to add the
     data from both
.gw.syncexecj["select sum size by sym from trade";`rdb`hdb;sum]

// Run the pre-defined functions - these are implemented to query the RDB and HDB as
    efficiently as possible

// Run a bucketed HLOC query, both as a string and in functional form
.gw.syncexec["hloc[2015.01.07;.z.d;0D12]";`hdb`rdb]
.gw.syncexec[(`hloc;2015.01.07;.z.d;0D12);`hdb`rdb]

// Run a count by sym across a date range, and add the results.
// Run both as a string and in functional from
.gw.syncexecj["countbysym[2015.01.07;.z.d]";`hdb`rdb;sum]
.gw.syncexecj[(`countbysym;2015.01.07;.z.d);`hdb`rdb;sum]
```



```
// Run a gateway query with a bespoke join function to line up results and compare
    today's data with historic data
.gw.syncexecj[(`countbysym;2015.01.07;.z.d);`hdb`rdb;{(`sym xkey select sym,
    histavgsize:size%tradecount from x 0) lj `sym xkey select sym,todayavgsize:size%
    tradecount from x 1}]

// Send a query for a process type which doesn't exist
.gw.syncexec["select count i by date from trade";`hdb`rubbish]

// Send a query which fails
.gw.syncexec["1+`a";`hdb]
```

5.1.2 Resilience

The gateway handles backend processes failing and restarting. To test it:

1. Manually kill one of the HDB processes (close the process on Windows, use the kill command on Linux or OS X)
2. Run one of the gateway queries which uses an HDB
3. Kill the remaining HDB process
4. Re-run the query- the gateway should return a failure error
5. Restart one of the HDB processes. To do this either run the correct individual line from the start script, or run the full start script.
6. Re-run the gateway query- it should be successful

Check the monitor for changes when killing and restarting processes.

5.1.3 Load Balancing

New processes can be dynamically added and they will register with the gateway which will start running queries across them. To test it, create 3 client q processes which run queries against the gateway as below. Note that the code below could be pasted into a q script and run for each client.

```
// open a connection
q)h:hopen `::30000:admin:admin
// function that will take 5 seconds to run on the HDB
q)f:{system $[.z.o like "w*";"timeout ";"sleep "],string x}
// function that will query the gateway
q)g:{(neg h)(`.gw.asyncexec;(f;x); `hdb); h[]}
// run the query, print the time
q)sendquery:{-1"query took ",(string (system"t g[",(string r),"]")%10*r:1+rand 5),"%
    of expected time";}
q)do[100;sendquery[]]
```


Each client is trying to run a query on the HDB which takes 5 seconds. There are 3 clients, and only 2 HDB processes sitting behind the gateway. Each query will therefore take between 5 and 10 seconds, depending on arrival time. As the number of clients increases, the average time will increase.

Assuming the environment variables are set up, a new HDB process can be started like this:

```
q torq.q -load hdb/database -p 31302 -U config/passwords/accesslist.txt -o 0 -
    proctype hdb -procname temphdb -debug
```

This will automatically connect to the gateway, and allow more queries to be run in parallel.

5.2 Examine the Logs

Each process writes logs to $KDBLOG. These are standard out, standard error, and usage logs. The usage logs are also stored in memory by default, in the .usage.usage table. The table can be used to analyze which queries are taking a long time, which users are sending a lot of queries, the memory usage before and after each query, which queries are failing etc.

5.3 Reports

The Reporter process has a set of default "reports" configured in $KDBCONFIG/reporter.csv. These are:

- A memory check which runs periodically against the RDB and emails an alert if the memory usage goes above a certain size
- A count check which runs periodically against the RDB and emails an alert if a certain number of updates haven't been received by a certain set of tables within a given period
- A date check which runs periodically against the HDB after end-of-day and raises an alert if the HDB date range isn't as expected
- An example end-of-day report which runs against the RDB at a specific time and produces a csv report of high, low, open and close prices per instrument and emails it
- The same example end-of-day report as above but running against the gateway which then forward it to the RDB

The config can be modified to change the reports that are run. Some example modifications would be changing the thresholds at which alerts are generated, how often

they are run, and what is done with the results. New reports can also be created. The report process will need to be restarted to load the new configuration.

5.4 Access Control

5.4.1 Adding Users

For simplicity each process is password protected using the file $KDBCONFIG/passwords/accesslist.txt file. This can be modified to have different access lists for each process. To add a new user, add their user:password combination to the file and either restart the process or execute

```
q)\u
```

within the process.

5.4.2 User Privileges

TorQ possesses utilities for controlling user access in the form of different user identities with different access levels. For more information on how to configure this, see the "Message Handlers" section in the main TorQ document.

