Track Port

# Legacy

The track\_port system has existed in various forms over a number of years. Here I am documenting how track\_port has been working since around 2006. The core of the system is a mysql database and a number of scripts used to maintain/query the database. The details of each of these components will be outlined and described in the following sections, but here is a short summary of each component.

The database consists of a number of tables, but there are three tables that are most important. The table transaction\_list contains information about every position (symbol, open info, close info, etc.). With this information, portfolio tracking is possible. The table port\_history keeps track of portfolio names and their historical values and cash positions. Finally, the table finance\_quote contains daily quote data for all tracked symbols.

There are three main scripts that are used to update/query the database. The script quote\_query is a perl script that runs during the market open and repeatedly looks up quote info for all symbols. It then stores this info in the database finance\_quote table. The script port\_edit.cgi is a perl script used to enter/modify information in the database transaction\_list table. It is accessed using a web query. IOW, this is how positions are opened and closed (as well as providing some modify capability). Finally, the script pull\_transaction\_report.py is a python script used to show tabular portfolio info using a web query.

## Database

### Table transaction\_list

#### Describe

mysql> describe transaction\_list;

+--------------+------------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+------------------+------+-----+---------+----------------+

| id | int(10) unsigned | NO | PRI | NULL | auto\_increment |

| fileportname | varchar(256) | YES | | | |

| symbol | varchar(32) | YES | | | |

| sector | varchar(32) | YES | | | |

| position | varchar(16) | YES | | | |

| descriptor | varchar(16) | YES | | | |

| shares | decimal(14,4) | YES | | 0.0000 | |

| open\_price | decimal(14,4) | YES | | 0.0000 | |

| open\_date | date | YES | | NULL | |

| closed | tinyint(1) | YES | | 0 | |

| close\_price | decimal(14,4) | YES | | 0.0000 | |

| close\_date | date | YES | | NULL | |

| expiration | date | YES | | NULL | |

| strike | decimal(14,4) | YES | | 0.0000 | |

+--------------+------------------+------+-----+---------+----------------+

14 rows in set (0.00 sec)

#### Additional Info

The field position can be “cash” or “long”. The former is used for deposits, withdrawals, dividends, adjustments, etc. Essentially, anything that involves a dollar amount as opposed to an actual position. The latter is used for a security position. The original intent was that this could be “long” or “short” depending on how the position was opened. However, short positions are denoted with negative share counts.

The field descriptor is used to describe the type of position for long positions, ie. stock, call or put. For cash positions, it can be “initial” or “intermediate”. The former will set the date when the portfolio was created.

All the other fields are self-explanatory.

One further note regarding the field sector, dividends are marked as such by setting sector to “dividend”. By doing so, it is possible to include dividends as part of a position’s overall return.

### Table port\_history

#### Describe

mysql> describe port\_history;

+--------------+------------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+------------------+------+-----+---------+----------------+

| id | int(10) unsigned | NO | PRI | NULL | auto\_increment |

| date | date | YES | | NULL | |

| fileportname | varchar(256) | YES | | | |

| total | decimal(14,4) | YES | | 0.0000 | |

| cash | decimal(14,4) | YES | | 0.0000 | |

+--------------+------------------+------+-----+---------+----------------+

5 rows in set (0.00 sec)

#### Additional Info

Every day the market is open, a new row is created for each port with that date. The total is the total value of the portfolio (including cash). The cash is the amount of cash in the port on that date. This is the only place that cash is tracked. In retrospect, it would have been better to create a cash position for each portfolio in the transaction\_list table.

### Table finance\_quote

#### Describe

mysql> describe finance\_quote;

+------------+------------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+------------+------------------+------+-----+---------+-------+

| symbol | varchar(32) | NO | PRI | | |

| name | varchar(32) | YES | | NULL | |

| last | decimal(14,4) | YES | | 0.0000 | |

| high | decimal(14,4) | YES | | 0.0000 | |

| low | decimal(14,4) | YES | | 0.0000 | |

| date | date | YES | | NULL | |

| time | time | YES | | NULL | |

| net | decimal(14,4) | YES | | 0.0000 | |

| p\_change | decimal(6,2) | YES | | 0.00 | |

| volume | int(10) unsigned | YES | | 0 | |

| avg\_vol | int(10) unsigned | YES | | 0 | |

| bid | decimal(14,4) | YES | | 0.0000 | |

| ask | decimal(14,4) | YES | | 0.0000 | |

| close | decimal(14,4) | YES | | 0.0000 | |

| open | decimal(14,4) | YES | | 0.0000 | |

| day\_range | varchar(64) | YES | | NULL | |

| year\_range | varchar(64) | YES | | NULL | |

| eps | decimal(14,4) | YES | | 0.0000 | |

| pe | decimal(14,4) | YES | | 0.0000 | |

| div\_date | date | YES | | NULL | |

| dividend | decimal(14,4) | YES | | 0.0000 | |

| div\_yield | decimal(14,4) | YES | | 0.0000 | |

| cap | decimal(20,4) | YES | | NULL | |

| ex\_div | date | YES | | NULL | |

| nav | decimal(14,4) | YES | | 0.0000 | |

| yield | decimal(14,4) | YES | | 0.0000 | |

| exchange | varchar(32) | YES | | NULL | |

| success | tinyint(1) | YES | | 0 | |

| errormsg | varchar(40) | YES | | NULL | |

| method | varchar(32) | YES | | NULL | |

+------------+------------------+------+-----+---------+-------+

30 rows in set (0.00 sec)

#### Additional Info

Most of this data is not capture anymore as quote services have evolved over the years.

## Scripts

### Perl quote\_query

#### The Main Loop

Call db\_get\_symbols to get two lists (list\_symbols and list\_options).

Initialize finance\_quotes hash to be empty at the start of the loop.

If we have anything in list\_options, call get\_quoteoption\_data() to start filling in finance\_quotes.

Call get\_quote\_data1() to fill in more finance\_quotes entries.

There is some debug code that saves finance\_quotes to a file.

At this point we are ready to start db operations:

1. Lock finance\_quote table.
2. Delete finance\_quote table data (on first loop iteration).
3. Loop over list\_symbols and replace entries in finance\_quote table.
4. Loop over list\_options and replace entries in finance\_quote table.
5. Perform finance\_quote table commit.
6. Unlock finance\_quote table.

#### Function db\_get\_symbols

Essentially we query transaction\_list to create a list of symbols and a list of options. The symbol query has the following criteria:

* position in (‘long’, ‘short’)
* descriptor = ‘stock’
* not closed
* symbol <> ‘^DJI’

We use a UNION to include symbols from ticker\_symbols table (again symbol <> ‘^DJI’).

One special case to consider. We strip out any symbols that are 5 characters where last character is ‘X’ if fetch\_mf is False.

Building the list of options uses a query that selects:

* position in (‘long’, ‘short’)
* not closed
* descriptor in (‘call’, ‘put’)

Then we have to build each option as a symbol using symbol, descriptor, strike, and expiration.

TODO: put an example option symbol here.

#### Function get\_quoteoption\_data

The API is simple, pass in the list of option symbols and a pointer to the finance\_quote hash, fill in the hash with quote data for each option.

This should be a total re-write.

#### Function get\_quote\_data1

Here is what I have in the function header:

# Fetches are done using finance::quote fetch method. The difference between this

# function and get\_quote\_data is that this method works better when finance::quote

# returns a mix of good and bad data on a symbol basis. Empirically, here is what I

# observed. Assume we are fetching data for 500 symbols:

# o After the first fetch, anywhere from 150-300 symbols would have bad data.

# o Subsequent fetches would get a mix of good and bad data, but not the same symbols

# each time.

# o Some symbols seemed resistant to getting good fetches.

#

# Given that, here is the proposed algorithm:

# 1. Attempt a fetch of all symbols.

# 2. Loop through each symbol and determine if fetch was good or bad.

# a. If bad, delete that symbol from the fetch hash.

# b. Save the symbol for the next iteration.

# 3. Merge the resulting fetch hash into the accumulate fetch hash.

# 3. Delete bad symbols from the original list to be fetched and goto 1.

# 4. Repeat the above loop, N times or exit the loop if there are no bad symbols.

# 5. If there are still bad symbols, use aq method to fetch and fill accumulate

# fetch hash.

#

# Three parameters are passed:

# p\_finance\_quotes -- Pointer to the accumulate fetch hash (assume it was emptied before the call).

# p\_list\_symbols -- Pointer to list of symbols to fetch (NOTE: ^DJI is added to this list after aq fetch).

# p\_hash\_put\_stats -- Pointer to put\_stats hash.

This should probably be a total re-write.

### Perl port\_edits.cgi

### Python pull\_transaction\_report.py