Track Port

[1. Legacy 1](#_Toc54109072)

[1.1. Database 2](#_Toc54109073)

[1.1.1. Table transaction\_list 2](#_Toc54109074)

[1.1.1.1. Describe 2](#_Toc54109075)

[1.1.1.2. Additional Info 2](#_Toc54109076)

[1.1.2. Table port\_history 3](#_Toc54109077)

[1.1.2.1. Describe 3](#_Toc54109078)

[1.1.2.2. Additional Info 3](#_Toc54109079)

[1.1.3. Table finance\_quote 3](#_Toc54109080)

[1.1.3.1. Describe 3](#_Toc54109081)

[1.1.3.2. Additional Info 4](#_Toc54109082)

[1.2. Scripts 4](#_Toc54109083)

[1.1.4. Perl quote\_query 4](#_Toc54109084)

[1.1.4.1. The Main Loop 4](#_Toc54109085)

[1.1.4.2. Function db\_get\_symbols 4](#_Toc54109086)

[1.1.4.3. Function get\_quoteoption\_data 5](#_Toc54109087)

[1.1.4.4. Function get\_quote\_data1 5](#_Toc54109088)

[1.1.5. Perl put\_db\_quotes 5](#_Toc54109089)

[1.1.5.1. The Main Loop 6](#_Toc54109090)

[1.1.5.2. Function db\_parse\_transactions 6](#_Toc54109091)

[1.1.5.3. Function build\_fq\_hash 6](#_Toc54109092)

[1.1.5.4. Function create\_transaction\_report 7](#_Toc54109093)

[1.1.5.5. DB Operations on transaction\_report 7](#_Toc54109094)

[1.1.5.6. DB Operations on port\_param 8](#_Toc54109095)

[1.1.5.7. DB Operations on port\_history 8](#_Toc54109096)

[1.1.6. Perl port\_edits.cgi 8](#_Toc54109097)

[1.1.7. Python pull\_transaction\_report.py 8](#_Toc54109098)

[1.1.8. Perl port\_track.cgi 8](#_Toc54109099)

# 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.

In addition, the script port\_track.cgi is used to effectively generate a pull\_transaction\_report URL. Essentially, you can multiply select fileportnames and specify a few URL GET parameters. The beauty of this script is that it queries the database for all files as well as ports within each file.

## 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 put\_db\_quotes

#### The Main Loop

Call db\_parse\_transactions(). More details below, but this will parse transaction\_list and port\_param tables.

Call build\_fq\_hash(). More details below, but this will create a hash of finance\_quote data parsed from finance\_quote table.

Tease out $quote\_date from finance\_quote data for ^GSPC.

Call create\_transaction\_report(). More details below, but this will update port\_param table and transaction\_report table. The latter is not really used anymore.

After calling all of these functions, there are a series of DB operations performed. Essentially, there are 3 DB operations that touch the tables transaction\_report, port\_param, and port\_history:

1. drop/insert/commit transaction\_report
2. truncate/insert/commit port\_param
3. delete\_rows/insert/commit port\_history

#### Function db\_parse\_transactions

This function creates 3 data structures. Each of the structures is passed in as a pointer to the function and the functions modifies them in place.

@parsed\_transactions is a list of hashes containing all transaction\_list data (including options). It is initialized to an empty hash. There are three queries made to transaction\_list (one is specific to esop descriptors that are no longer used). We query transaction\_list for regular (long, short) positions:

* position in (‘long’, ‘short’)
* descriptor == ‘stock’
* not closed

These rows are fetched and we fill in the hash with file, port, symbol, label, sector, date (open), purchase, qty, id.

Then we query transaction\_list for options positions:

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

These rows are fetched and we fill in parsed\_transactions hash with file, port, sector, date (open\_date), purchase (open\_price), qty, id. We also create the option symbol using descriptor, expiration and strike information.

%port\_params is a hash containing data from the port\_param table.

@list\_cashonly\_ports is a list of ports that are cash-only (no open positions).

#### Function build\_fq\_hash

This is actually buried inside a watchdog while loop such that it looks at finance\_quote data and simply delays 60 seconds if it doesn’t find anything valid for ^GSPC.

The function has two parameters, each is returned from the function. @list\_fq\_fields is a list of fields available in the finance\_quote table. %hash\_fq is the hash containing the actual finance\_quote table data. It is a two level hash where the first index is the symbol and the second index is the field name.

Querying the database for the fields is done using:

DESCRIBE finance\_quote;

Querying the finance\_quote data is done using:

SELECT \* FROM finance\_quote ORDER BY symbol;

#### Function create\_transaction\_report

This is just a straight function call after initializing the list @transaction\_report. The function has 5 parameters, all of which are passed as references.

Here is the function header comment pulled from the code:

# create\_transaction\_report

# Input parameters:

# p\_parsed\_transactions -- Pointer to the parsed transactions read from input file.

# p\_list\_cashonly\_ports -- Pointer to the list of cash-only ports.

# p\_port\_params -- Pointer to hash containing port parameters.

# p\_hash\_fq -- Pointer to hash finance quote info.

# Output parameters:

# p\_transaction\_report -- Pointer to list of hashes containing transaction report data.

#

# Description

# There are two passes through the parsed\_transactions/options list. The first pass has

# two objectives:

# 1) Total each portfolio (according to fileportname) and store in port\_params hash.

# 2) As portfolios are built, create a mirror image portfolio that combines transactions

# that involve the same symbol. These are appended to the parsed\_transactions list

# as needed and a uniquified fileportname is created for each by appending '\_combined'

# to the port name.

# The second pass through the parsed\_transactions/options list has a single objective:

# 1) Create transaction\_report data for each transaction/option in the list.

This description is accurate and fairly detailed. The key there is the two-pass algorithm where the uniquified “combined” ports are created.

The interesting thing is that most of the work done by this function is not really needed anymore. Instead that work is done when pull\_transaction\_report URL is called.

#### DB Operations on transaction\_report

Lock the table ( LOCK TABLES transaction\_report WRITE; ).

Delete all the current rows ( DELETE FROM transaction\_report; ).

Insert new rows (execute\_query\_transaction\_report()). I don’t want to get into these details, they will be different in python. But effectively we build an INSERT INTO transaction\_report …; query and execute it.

Commit the changes.

Unlock the table ( UNLOCK TABLES; ).

#### DB Operations on port\_param

Truncate the port\_param table ( TRUNCATE TABLE port\_param; ).

Insert new rows (execute\_query\_port\_param()).

Commit the changes.

#### DB Operations on port\_history

Delete rows from port\_history (only for the current date).

Insert new rows (execute\_query\_port\_history()).

Commit the changes.

### Perl port\_edits.cgi

### Python pull\_transaction\_report.py

### Perl port\_track.cgi