## Database Design Question README

This is a README to go over the logistics of how the database was configured and how it can be emulated. Unless otherwise specified, all files that I mention here should be put in the same directory which you can use as the workspace.

### Setup

The design is elaborated on in the “Portfolio positions database design” page. The database itself is relational and uses SQLite to be precise. This was used since it is simple to set up and emulate and I can provide the database file directly. The SQLite database simply resides as a file on the user’s local machine so there is no need to have connection to a remote host or any configurations at all. The file which I have sent called “portfoliosdb.db” is the full database file.

This database can be loaded after installing SQLite. I personally installed sqlitestudio which has an interface to view/query/update the database. I did this on Windows and but it can be downloaded for any OS on this link <http://sqlitestudio.pl/?act=download>.

After installing sqlitestudio (or any SQLite distribution), you can open the SQLiteStudio.exe (on Windows) application and choose to “Connect to a database”. On this panel, browse to and select the location of the portfoliosdb.db. You should see the portfoliosdb database in the navigator to which you can browse and view any information related to the database such as indices, views, table schemas, associated DDLs, etc. You can see the design, implementation, and marketvalue\_positions\_view as explained int the design document.

### Loading data

To load the data, you can use 2 files which are included: securitiesfile.csv and tradesfile.csv. This is a comma separated file in which you can specify the data to be loaded into each table as a comma separated list in the same order as the columns.

Example line in securitiesfile.csv to insert a row into the securities table:

US1111111114,15.0,30.0

Example line in tradesfile.csv to insert a row into the trades table:

17,US1111111114,0,250.00,ABC

The loader script is called “portfoliodb\_loader.py”. This script will look for the database, securities csv file, and trades csv file in the same directory and try to load the data from the files into the database. I ran it by running the following under Python 2.7

>>> python portfoliodb\_loader.py

### Web application

There is also a simple web application built to be able to display the marketvalue\_positions\_view View in the browser through HTML. It has a feature to filter the positions that have market value above the given (input) threshold.

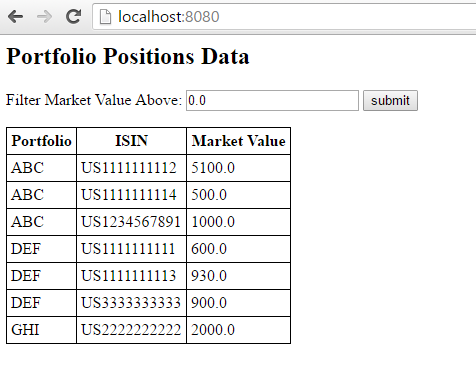
The script to start the web application is “run\_web\_app.py”. This script has dependencies on the web.py module and the HTML.py modules which are both third-party software that has facilitated this project. The download web.py, you can go to <http://webpy.org/install>. I personally used this link, <http://webpy.org/static/web.py-0.37.tar.gz>. After downloading the zipped file, I unzipped and extracted the “web” directory to the workspace directory (same directory as the script). Therefore, there is no installer to run. just take the “web” directory and copy that to where the run\_web\_app.py script is. To get HTML.py, I downloaded the zipped file from this page <http://www.decalage.info/en/python/html#attachments>. Once again, there was no installer to run. You only need to copy the HTML.py file to the workspace where you are running the script.

With all the dependencies in place, there is one more step before starting the web application. The HTML file that will be rendered has to be in place (called index.html). In the workspace directory, create a directory called ‘templates’ and put the provided index.html file in there. The script will use this when rendering the view.

Now, to start the web application, run the script which starts the application under port 8080 on your local host (using web.py). I ran it under Python 2.7 using

>>> python run\_web\_app.py

Once started, go to localhost:8080 in the browser. You should see a view similar to the following. The is a direct rendering of the marketvalue\_positions\_view as an HTML table (using the HTML.py module).



You can use the market value filter to only show the positions with market value above the input. This is where the filter explained in the design document comes into play. Below is an example, on filtering for market value > 900:

