Database Structures for Financial and Risk Analysis

[Nematrian website page: DatabaseStructureIntro, © Nematrian 2015]

Given the desire for regulators, analysts, fund managers, risk managers and traders to store and analyse more and more historic data it is perhaps surprising that standards have not yet been established for the manipulation and analysis of historic time-ordered data.

This contrasts with the de facto development of *Structured Query Language* (SQL) and relational database technology as the standard for databases more generally.

Go back c. 30 years and there was no standard way of analysing and storing data – different hardware (and software) vendors developed their own proprietary database approaches that were often cumbersome to use. What was needed was a more standardised way of storing data (hence relational databases) and for querying this data (hence SQL).

However SQL-style relational databases are not ideal for storing and analysing time-series data, see Sentance (2008), or for that matter data that has any type of inherent 'ordering'. The assumption underlying a relational database is there is no meaning ascribed to any particular row in a given relational database. The same data would have exactly the same meaning if it appeared at the start of the table or at the end of the table, although there might be some difference in processing time in cases where we sequentially search for data from one or other end of the database. This is because SQL is orientated to deal with mathematical "sets" of data and on the "relations" between them, and such sets do not by themselves include ordering. Thus the set {3, 5, 2} is the same as the set {2, 3, 5}, because both have exactly the same constituents.

For some simple time-series based analyses, relational databases work fine. For example, suppose we want to load a particular time series for a particular stock from a relational database. The database (say, called HISTORY) might have several fields such as:

NAME
DATE
VALUE1
VALUE2
...

To query this data to return the desired time series we might:

```
SELECT DATE, VALUE1
FROM HISTORY
WHERE NAME= "EXXON"
ORDER BY DATE
```

The challenge arises when we want to process this data. Suppose that VALUE1 is in one table and refers to price and VALUE2 is in a different table and refers to volume (no of shares traded), and we want to know the value traded, i.e. in some sense VALUE1 times VALUE2. We then need a more complicated query that in effect multiplies these two vectors together. Calculation of risk statistics such as historic volatilities and correlations is more complex still. Indeed these sorts of calculations can become so complicated that it becomes impractical to incorporate such functionality within a standard SQL framework.

Specialist database vendors have thus created proprietary database systems that are better suited to carrying out these more complicated vector manipulations (e.g. allowing users to include 'stored procedures' within the database, each one of which able to manipulate the data in the database in an essentially arbitrary fashion). High-performance database vendors also typically unbundle their databases into two parts – one part that is in-memory, say for intra-day data, and one part that is copied down to disk, say overnight. This approach is driven the fact that disk-speed improvements have trailed far behind CPU speed improvements. Alongside these approaches we have the likes of Microsoft developing tools such as LINQ that allow more 'native' manipulation of database-sourced data directly in standard computer programming languages.

Whilst the Nematrian website is not aiming per se to provide large scale database functionality, its data requirements do have some similarities with those of time-series orientated databases. It thus incorporates in its internal workings approaches that are suited to analysing time series data as well as those applicable to more traditional database structures.