



Brian Sentance

# SQL and Historic Data — Mixing Oil and Water?

“History is merely a list of surprises. It can only prepare us to be surprised yet.”

**W**hether or not current market conditions lead you to endorse the above quote,<sup>1</sup> history and its analysis is a vital building block within financial markets. Given the desire by regulators, risk managers, and traders to store and analyze more and more historic data, it is perhaps also surprising that no standards have been established for the manipulation and analysis of historic time-ordered data. Structured Query Language and relational technology continue to dominate the database market, while a variety of specialist solutions deliver high performance but often require expensive knowledge of proprietary tools and techniques. Business users need ease of access and performance levels above those offered by standard relational database solutions, but technologists are extremely reluctant to head down the proprietary route. So what is the problem and what is to be done?

## Database History Repeats Itself

Looking at the variety of competing time-series technologies around at the moment, it is probably worth looking back 30 years to see why rela-



tional database management systems (RDBMS) and Structured Query Language (SQL), the language typically used to access them, were created in the first place. At that time, there was no standard way of storing and analyzing data, with hardware vendors developing their own proprietary database solutions for the “tin” (main-frame computers to those of you who aren’t as old as me) that they sold to clients. These proprietary solutions were often difficult and cumbersome to use and required specialist and expensive expertise to manage. With the advent of

more portable hardware, what was needed was a more standardized way of storing data (leading to the relational database) and a more standardized way of interrogating it (leading to the SQL programming language). While hardware is (mostly!) kept separate from software these days, there is a reasonable parallel between why relational databases and SQL came about and the current state of affairs with the competing proprietary solutions for the time-series space. Standardization of time-series storage and access would increase productivity and reduce costs of

analyzing high-volume/complex data within financial markets.

## Data Analysis—What’s Wrong with SQL and Relational Storage?

Answer: Not a lot, for a lot of things. I would propose the following reasons for this: SQL is an industry standard (however hard the vendors add proprietary extensions!); the technology is mature and robust; and technologists find SQL easy to understand and implement. Additionally, once you have learned it, you will be able to reuse this knowledge over a wide range of databases and database-based applications. Put another way, given that it is ubiquitous in the market, you might as well learn (some of) it if you are able. The only minor rub of this is that being a general language, it can soon become quite complex to do something that to a business user appears on the surface to be quite simple, but more on that below.

## SQL and Time-Ordered Data

One area where SQL still has issues is in dealing with structured and time-ordered data. SQL is oriented to deal with mathematical “sets” of data and the “relations” between them (hence, “relational” databases). However, the nature of a set is that it has no order. So in this way SQL is not particularly well designed to deal with time-ordered, sequential data such as tick histories or more complex (but less voluminous) historic correlation or implied volatility surface data. Put another way, SQL has no intrinsic knowledge of how one row of a resultant dataset is related to another.

To illustrate this, let’s imagine that we want to load a particular time series for a particular stock from a relational database. Below is a very simple table, called History, used as an example of how such time-series data for a number of instruments could be stored:

History		
Column	Data Type	Description
InstTicker	CHAR(12)	Instrument name
SeriesName	CHAR(12)	Time-series name
Date	DATETIME	The date associated with a point in the time series
Value	DECIMAL	The date associated with a point in the time series

And here is how you could query this data (“Load a Series” type functionality) for a particular time series for a particular stock:

```
SELECT    DATE, VALUE
FROM      HISTORY
WHERE     ITEMID = "IBM.N"
AND       SERIESID = "TRADEPRICE"
ORDER BY  DATE
```

The above is quite easy, even for nontechnical specialists, and might well be all you need to load your data up into your preferred mathematical package to start manipulating the time-series vector. However, doing something more complicated directly in SQL, such as multiplying the “TRADEPRICE” by another time series such as “TRADEVOLUME” requires a lot more work (and, hence, query time), with multiple “joins” (links between tables and result sets) being required to achieve this kind of vector multiplication.<sup>2</sup> Doing something only slightly more complicated, such as a historic volatility calculation, is more complicated to implement and expensive to run.<sup>3</sup> Further complication is added when the business user wants data that is time-aligned and interpolated to a different frequency, or with missing data filled according to some reasonably complicated rule set. Soon these kind of manipulations become too much to do in SQL and, as a result, many end-users typically revert to manipulating the data in spreadsheets to achieve what they need. It is then that the desktop spreadsheet risks being used as a database and operational risk problems usually follow. Technologists would ideally adopt a more centralized approach and opt to implement this logic in what is often referred to as the “business layer” (the bit of their system where the business logic is managed). This is fine but tends to be very time-consuming (hence, expensive) and often requires reworking for each (slightly different) business problem as it

crosses up. This is usually because the company they work for does not have the time or R&D appetite to design and implement a full-fledged generic “language” to express and execute these

business problems. That’s where the presence of standards would help, as organizations would then have the “language glue” they need and ideally could just add their own extensions as they felt necessary.

This difficulty in using SQL with historic data has led to the proliferation of proprietary languages (that are good at manipulating vectors) among time-series vendors—for example, those from KX Systems<sup>4</sup> and from my own company Xenomorph.<sup>5</sup> Many of the vendors in this area provide an SQL interface on the data stored in their database systems, but to do something more complicated they provide tools and languages that make vector manipulation easy and quick (at the cost of specialization). Some business users are prepared to make the effort to learn these languages (productivity outweighing the cost of learning), but many technologists will stick with what they know, since they already know SQL and probably don’t have the time (or business incentive?) to learn another database language.

## Relational Storage and Time-Ordered Data

I think it is important to differentiate between the brief discussion of SQL with time-ordered data above and the relational storage of time-ordered data. Both go hand in hand to a large degree, but many proprietary database solutions do not use relational storage and provide some (often limited) form of SQL access. Let’s go back to the simple example of the History table to understand a basic problem that relational storage has with time-ordered data.

In order to retrieve the time series, the query must search through every row of the table (a full table scan) to evaluate whether it meets the filter criteria set (in this case, the rather undemanding request to retrieve trade price data for IBM) and then return the results found. As more data are added to the table, the query will take longer and longer to execute before returning the required dataset. While obviously the simple database design above can be “industrialized” to increase performance, increasing tick data volumes have led to billions of rows being needed, something that the original relational databases

were simply not designed for.

Expanding on this simple example of how relational, row-based storage struggles with time-ordered data, there are other related problems to face:

- Speeding up time-series access through indexing increases the database storage required due to the repeated columns required to index data.

- A data object (say a day's tick data, or a historic vol surface) should ideally be clustered together, but its data very often become fragmented within the rows and tables of the database.

- Too much searching is then required to obtain a complete or even just a partial image of the data object.

- Reconstruction of data objects from table rows is intensive and means that operational intelligence for dealing with those data objects generally has to be handled above the data layer in the business layer.

- As with SQL, relational storage does not lend itself to a clean representation of time-ordered and structured data.

- Relational databases were predominantly designed for low-frequency updates, and this flies in the face of the high-volume, simultaneous read/write levels demanded today.

A lot of the arguments put forward above are the same as the primary argument put forward a decade or more ago by the “object database” vendors—i.e., that it is inefficient to continually deconstruct then reconstruct your business object (a yield curve, correlation matrix, time series, etc.) into a row-based relational data model. The argument fundamentally still stacks up, particularly in the financial markets industry, where the simple (but ever faster!) storage and retrieval of data is still a primary business usage pattern.

## The Column Approach

The term “column database” seems to be gaining in (marketing?) popularity lately. What happened to the good old “time-series database” used over many decades by products such as FAME?<sup>6</sup> Even one of the larger relational database vendors, Sybase, seems to be benefiting greatly of

late from selling its own object-/column-/vector-based database solution Sybase IQ,<sup>7</sup> validating that relational databases are not the complete answer to everyone's problem, even for relational database vendors. The approach being taken to address many of the relational database storage issues outlined above is to organize your time-series and complex object data in columns rather than in rows. This has two main advantages: (1) the column data can be stored contiguously on disk, allowing for much faster read and write operations and (2) many SQL queries only return a few fields (date and value anyone?), so rather than having to read all of the row data off a disk you are able to only retrieve the data you ultimately need, speeding up access through reduced disk I/O. Additionally, this kind of approach can:

- eliminate redundant index data and, hence, save on disk space;
- keep the object data together, reducing overall table size for quicker location scans;
- allow read/write times to be optimized for speed, which is particularly crucial for capturing high-volume, real-time data;
- allow the query language that extracts data to be optimized with built-in awareness of structured column data for greater simplicity and speed;
- make it easier to apply security models at the data object level;
- enable data intelligence to occur closer to the data—again delivering extra performance benefit; and
- deliver performance gains by having dedicated write streams built into the database product.

While some relational database purists will be shaking their heads in despair at this point, the performance benefits for accessing certain types of time-series and structured data using the above approach are plentiful. For example, one European investment bank we have dealt with managed to reduce their VaR calculation down from over 12 hours using a relational solution to well under one hour using a column/time-series approach. Some newer players in the column database market, notably Michael Stonebraker of Vertica (and Streambase), believe that the archi-

tures used by the big database vendors are antiquated and need a fundamental rewrite.<sup>8</sup> Coming back to the relational, row-based world of the main database vendors, it would be a logical response to this criticism if they were to implement some form of (optional) column-based storage within their existing offerings (or buy-in column-based database technology as Sybase did with IQ, and as IBM did with Informix).

## A Standards-Based Approach to Time Series?

Speaking of the standards-based database vendors, many of them have long accommodated the need for the storage of nonstandard data (such as time series) by allowing binary data types in tables. This enables a series of data to be vectorized and stored as binary data within a column similar to the approach outlined above. This approach has many of the benefits of the simple column approach outlined above, including reduced storage and retrieval times, but the penalty is the loss of query flexibility, as the data is no longer transparent to SQL. Put a simpler way, this binary storage approach means that SQL can no longer be used to access the time-series data directly. Hence, packing and unpacking of the time-series data must take place at a higher layer outside of the database, and hence one of the benefits of the column solution above is lost.

However, some database vendors have introduced user-defined data types for table columns. In the case of the SQL Server 2005, from Microsoft, developers are able to do this within the .NET framework CLR hosted within the database. We took a look at this a while back and were unfortunately disappointed to find that the size of each datatype was limited to a level that did not make it practical for time-series storage. Microsoft has subsequently removed this restriction in the SQL Server 2008, which means that it is now worth another look, opening up the SQL Server to both the storage of complex data types and their access within the SQL language, particularly in combination with user-defined functions (UDFs) to increase the query power and functionality available in standard SQL to perform financial calculations. On the language

side of things, another interesting development from Microsoft is Language Integrated Query (LINQ).<sup>9</sup> LINQ allows querying to occur directly within the code of .NET programs, as opposed to having to provide quoted SQL statements via middleware functionality such as ADO.Net. Customizability of data types, combined with an easy way to extend an accepted database language, may prove to be a good route to explore for those seeking to utilize standards-based technology for these types of problems.

Taking some of the customization approaches outlined above, my own company, Xenomorph, has taken a detailed look at hooking our time-series database technology into the Microsoft SQL Server. What we have tried to take on board is our client's desire for standards-based database management tools; accessibility from SQL as well as from a vector-optimized language; and column database performance. Initial benchmark results are very promising, so we shall see whether our investment in being able to deliver high performance from a standards-based solution pays off over the coming months.<sup>10</sup>

## In-Memory Databases Head Toward the Grid

Many high-performance database solutions are currently based around an architecture that involves an in-memory database for daily tick data, with copying done “overnight” to a historic archive database stored on disk. This in-memory approach for capturing real-time data means that write speeds are not limited by the physical process of writing data to a disk, making it faster to store updates. Then, assuming that your overnight bulk copy is fast enough, you are all set for the next day. That said, I am not sure the concept of “overnight” really has legs anymore given the move toward 24-hour, global trading. It is probably also worth noting that this approach has been driven because of the fact that disk-speed improvements trail way behind CPU speed improvements over the past 25 years.<sup>11</sup> One obvious issue is ensuring that data are not lost should the holding server for the in-memory database go down. In addition, having separate historic stores makes it potentially more complex to retrieve

data that spans from now to some point prior to today—ideally, this fact should be hidden from the end-user.

Moving on from the single in-memory database approach, another key technology in the high-performance database market will be the data (aka *data fabric*) technology. A data fabric is a distributed in-memory cache for data, coming out of the need to avoid the “data-bound” computational grid,<sup>12</sup> and this type of cache will move out of its traditional home in high-performance computing (HPC) to play a big part in the construction of high-performance database and data management solutions (see the recent acquisition of Tangosol by Oracle for instance). Standards-based databases are designed for general use with low update frequencies (relative to those required in the financial markets); hence, you need the data fabrics to buffer the data and in order to give you some “behind-the-scenes” time for optimal disk-write performance that can meet the levels required by the market. In addition to the possibility of increased throughput, data fabrics also offer the benefit of being able to access the same data throughout an organization, where it is needed. With database replication being difficult to manage and maintain, this method of data “globalization” may well prove increasingly popular.

## Summary

From a language perspective, SQL is certainly here for the long term. Whether anyone will attempt to build a standardized (but workable!) time-series extension to it is difficult to say. Certainly, there is some interest in establishing a standard from academia,<sup>13</sup> but maybe some of the newer ways of accessing data (LINQ?) will offer the customization levels needed to properly integrate time-series and complex data objects. From a data management perspective, an ideal solution to high-performance analysis of time series would be to take the best of all worlds: the best storage layers, the best data models, the best data fabrics, and the best HPC technology for the analysis of all of this data. This pick ‘n’ mix approach is an option for some cash- and resource-rich organizations but requires real technical (and business!) expertise and a strong

commitment to support and keep up to date with the latest technology. With increased customizability being offered by the mainstream database vendors, and the rising trend toward data fabrics, grid computing, and virtualization, it will be interesting to see how proprietary time-series database vendors react. Watch this space!

Brian Sentance's company blog can be found at <http://xenomorph.typepad.com>.

## FOOTNOTES

1. Quote by American novelist Kurt Vonnegut (1922–2007).
2. For some more detailed SQL and time-series examples, see the white paper “So Where’s the Table?” at <http://www.xenomorph.com/downloads/whitepapers/sql/>.
3. See Appendix A of the above white paper for some examples of SQL implementing historic volatility.
4. KX Systems provide documentation on their Q language at <http://www.kx.com/q/d/primer.htm>.
5. An introduction to Xenomorph's approach to vector manipulation can be found at <http://www.xenomorph.com/downloads/whitepapers/ql-plus/>.
6. FAME is now part of Sungard; see <http://www.sungard.com/fame/>.
7. See Sybase IQ news article at [http://news.yahoo.com/s/cmp/20080301/tc\\_cmp/206901052](http://news.yahoo.com/s/cmp/20080301/tc_cmp/206901052).
8. See Michael Stonebraker's blog at <http://www.databascolumn.com/2007/09/stonebraker.html>.
9. See some background on LINQ at <http://msdn.microsoft.com/linq>.
10. See case study at <http://www.microsoft.com/casestudies/casestudy.aspx?casestudyid=4000001637>.
11. See “A Brief History of Database RAM Storage” ([www.dba-oracle.com/t\\_history\\_ram.htm](http://www.dba-oracle.com/t_history_ram.htm)).
12. See my Wilmott March 2007 article “Cluster Wars—Scaling-Up Derivatives,” which can be found at <http://www.xenomorph.com/news/inthepress/2007/mar/wilmott/>.
13. For example, Streambase is trying to commercialize a “standard” for querying data streams; see <http://blogs.streamsql.org/>.