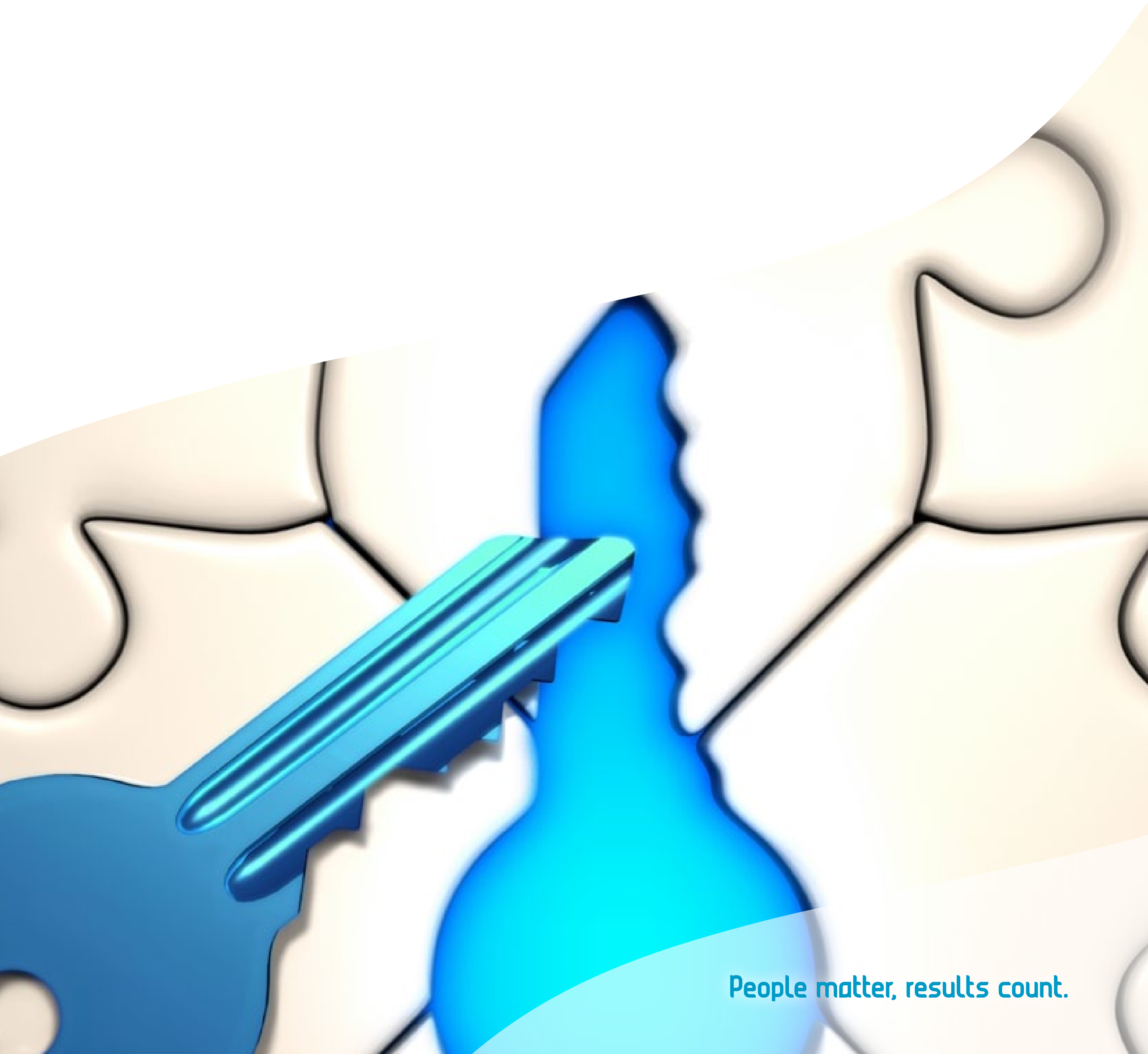
Enhancing Time Series Data by Applying Bitemporality

It's not just what you know, it's when you know it



Contents

1 Introduction to Time Series Data

While it was sufficient in the past to model the world based on the current state of things, it is not so simple in today's world of information and extensive data mining.

The definition of time series is a "…sequence of data points, typically measured at successive time instants spaced over uniform time intervals."[1] A simpler and arguably more accurate definition would be that *all data is time series*.

Sometimes in traditional relational databases we only see the current state of an entity. While it was sufficient in the past to model the world based on the current state of things, it is not so simple in today's world of information and extensive data mining. New technologies have given us the opportunity to take advantage of past information to make real-time analytical decisions and forecast the future.

The question that a business should be asking itself is not why we lost 500 customers last month, but how many customers we will lose this month and how to reduce that number. By analyzing the history of states of a given entity, the business will not only generate higher quality reports, but also make smarter business decisions.

Consider the example of a car manufacturer and its tire department.

Dept ID	Dept Head	Employee Number	Number of Projects
123	Martin Black	100	3

The table above gives us very limited information for analysis. All we can infer is that Martin is in charge of department, he has 100 subordinates and 3 active projects.

Now let's make this into a time series:

Date	Dept ID	Dept Head	Employee Number	Number of Projects
2010.03.01	123	Jason Brown	30	1
2011.01.15	123	Jason Brown	40	1
2012.01.04	123	Jason Brown	95	2
2012.03.15	123	Martin Black	100	3

[1] http://www.datastax.com/solutions/time-series-data

By this simple transformation we put ourselves in a position to answer so many more questions than before. We can correlate this seemingly static information with any number of other time series data we have gathered. Is there any correlation between customer complaints and the number of employees in the department? Has employee turnover increased/decreased since 1/12010 and what could be the cause of it? How does the growth of the company correlate with the growth of employees in this department?

The number of questions that we can answer seems to grow exponentially by using time series data that we have collected. Modern database technologies help us answer the questions in real time and perform the analytics needed to solve any problems that arise.

A few more concepts are important to understand about time series data. One is that the data is divided into two categories: **periodic** and **aperiodic**. Periodic data is data that comes in on pre-defined intervals of time such as monthly, daily, hourly or quarterly. Examples are the Consumer Confidence Index in finance or daily weather forecast updates. Aperiodic data does not come in pre determined intervals and can come in at any time. Examples include stock quotes throughout the day or solar storm eruptions.

The concept of bitemporality allows us to consider not only the recorded states, but also the time intervals for which the states were valid. The simplest way to express bitemporality is to add another timestamp to the database—valid time.

Quote

Transaction Time	Valid Time	Ticker	Bid	Ask
10:01:00.100	10:01:00.050	C	37.58	37.61

A little more advanced way is add a few more timestamps:

Quote

Start Transaction Time	End Transaction Time	Start Valid Time	End Valid Time	Ticker	Bid	Ask
10:01:00.100	10:01:00.100	10:01:00.050	10:01:00.070	C	37.58	37.61

Whichever method is chosen depends solely on the complexity of the problem and the business needs.

2 Why Should You Use Bitemporality?

Incorporating valid and transaction times and monitoring the difference between them helps an institution avoid making a decision with stale information.

There are many advantages to introducing bitemporality to your data. Perhaps the biggest one is that it allows the business to estimate the accuracy of the data. Information is money, and being able to know something fast means more opportunities for the business and a better bottom line. A high-frequency trading entity might care about the time a specific trade happens, but it also cares greatly when it knew that this trade happened. If his information is few hundred milliseconds old, then what chance does this entity have to compete against other high frequency firms that receive the information few microseconds after the fact? Incorporating valid and transaction times and monitoring the difference between them helps an institution avoid making a decision with stale information.

Bitemporality is an extremely effective way to make smarter investment decisions for the business. A rating company issues a rating for a corporation based on reported financial information by that company. What if the company updates its past financial reports sometime in the future? The rating company needs to go back and re-evaluate its ratings based on the newly available information. If a particular hedge fund uses the rating information to make an investment decision, it will benefit greatly from knowing both the valid time frames for a specific rating as well as the published transaction times, if only to know whether the rating reflects the most current information.

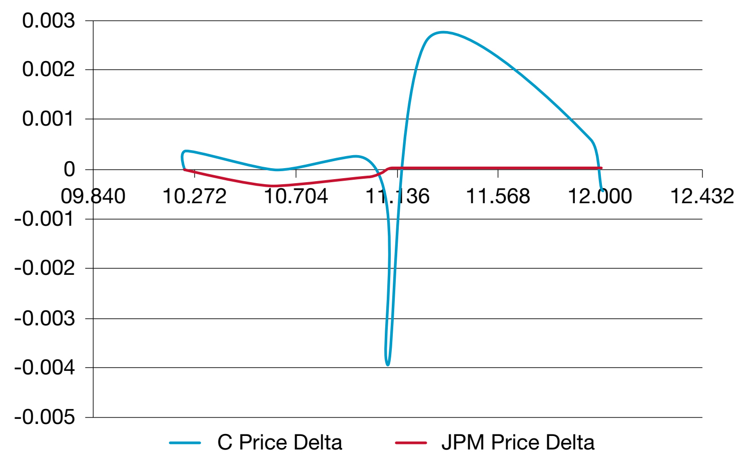
In addition, adapting bitemporality can help a business identify and understand any problems that have occurred. If a decision was made based on some information in the past and the results are not as expected, there is always a way to regenerate the state of things in the past and to analyze why the decision was made. Was it a wrong decision to begin with or was it the right decision based on the wrong information? How can such decisions be avoided or fixed early in the process?

2.1. Example 1: Bitemporality in Trading

Let us examine one simple example of a trading strategy. Let us assume that we are backtesting a pairs trading strategy and our positions are being opened when two particular equity instruments diverge more than a certain dollar amount. This example is oversimplified but will be good enough for our purpose.

C					JPM				
Valid Time	Bid	Ask	Bid Size	Ask Size	Valid Time	Bid	Ask	Bid Size	Ask Size
10:01:10.233	37.83	37.85	100	200	10:01:10.111	45.18	45.19	100	200
10:01:10.252	37.85	37.86	200	100	10:01:10.599	45.16	45.18	200	300
10:01:11.100	37.7	37.72	100	200	10:01:11.056	45.16	45.17	400	200
10:01:11.257	37.8	37.81	300	700					
10:01:11.901	37.83	37.85	200	100					
10:01:12.010	37.82	37.83	400	300					

Exhibit 1: Price change over time for C and JPM



What trading decision would we make based on the above information if our trading pair is Citigroup and JP Morgan? We can see that at 10:01:11.100 the prices for C dropped more than expected, so in a naïve world we would buy C and short JPM, and wait for them to converge back before we unwind the position.

In the real world there is not nearly enough information to make this kind of decision, because we do not have any indication of when we know the quotes. If our system receives the quote at 11:01:11.300 then the state of the world will have already changed and that information will be useless. Therefore, in the example above it's absolutely essential to have both time stamps (Valid and Transaction). If there are two time stamps available, we can see if our system is behind in processing the information. In the best case we can perform a thorough analysis and find the root cause of our system latency and infer the speed of the competition.

2.2. Example 2: Bitemporality in Insurance

Now let's examine a fictional insurance company XYZ, which decided to adapt the bitemporality approach. Three main data entities that come to mind are Customer, Policy and the Linkage between the two. The potential tables for XYZ insurance could look something like this:

Customer

Transaction Date	Valid Start Date	Valid End Date	Cust Id	Cust Name	Cust Type	Cust Address
2012.02.01	2012.01.01		19299	John Smith	Private	122 Long Pond Rd, Brooklyn, NY 11222
2012.03.05	2012.01.01	2012.03.04	19299	John Smith	Private	122 Long Pond Rd, Brooklyn, NY 11222
2012.03.05	2012.03.04		19299	John Smith	Private	10 Wilson Ave, Staten Island, NY 10306

Customer Policy Link

Transaction Date	Valid Start Date	Valid End Date	Cust Id	Policy Id
2012.02.01	2012.01.01	2012.06.01	19299	120
2012.03.04	2012.03.01	2012.08.01	19299	100
2012.06.01	2012.01.01	2012.05.01	19299	120

Policy

Transaction Date	Valid Start Date	Valid End Date	Policy Id	Insurance Type	Policy Amount	Policy Price
2012.01.01	2012.01.01	2013.01.01	120	Car	200,000	300
2012.02.01	2012.05.01	2013.05.01	100	Boat	300,000	100
2012.03.01	2012.03.01	2013.01.01	120	Car	200,000	400
2012.05.01	2012.05.01	2013.05.01	120	Car	200,000	420

In our simple example we can see how the customer entity 19299 changes through time. We can see that on 2012.03.04 the customer purchased boat insurance and, coincidently, his address changed on or about the same date. By analyzing more data around that geographical area, the insurance company may conclude that the area has many waterfront properties and generate a marketing campaign for boat insurance for all customers in that area.

We also see that 2012.05.01 the customer dropped the auto insurance and at the same time premiums for that type of policy went up in the system. It is not sufficient to make conclusions by analyzing just one customer, but if we have had a spike in drops during the premium change and the trend is increasing, then maybe we should reevaluate our pricing in comparison to the competition.

We also see that our system only knew about policy cancelation on 2012.06.01. The question that the company should be asking itself is why was it added 1 month after the insurance was terminated? This exposes a possibility that our data loading processes may be delayed for some reason and this could really impact the business adversely. All of the information in this example is quite limited. If we had something like claim information in bitemporal format, there would be many more smart business decisions that this insurance company would be able to make. For example, the company could:

- Estimate the number of future potential claims, based on the historical information.
- Price out different policies to maximize revenues and minimize expenses due to claims.
- Identify the most risky customers and their profiles.
- Identify the most profitable geographical areas
- Identify how the **effective cost per policy** evolved over time and what factors may have contributed to its changes.

3 Challenges to Overcome

If we just add date columns this would not only increase storage costs, but would also have architectural implications for the system design.

As useful as time series and bitemporality are, they entail a certain overhead. Each new state requires an addition of a row, and the more states changes happen, the more rows we add. As we can see from our previous employee/department example, just by adding the date to the table we have increased the size of the table by a magnitude of four. Add bitemporal capabilities with valid start times and valid end times and the table can easily grow 8 to 10 times bigger than the original one.

Yet in many queries and reports, our main interest would be just one record—the current state. The problem is that with time series database it is prohibitively expensive to remove rows. As the columns are stored contiguously removing a row would require copying the rest of the columns to fill the gap.

Therefore, if we just add the date columns this would not only increase our storage costs, but would also have some architectural implications for our system design. In this particular case we could employ at least a few different solutions to speed up queries. One solution is applying an index to the employee number column. This would speed up the queries significantly and allow us to keep our dates without a significant performance hit.

Another and perhaps more elegant solution would be to have the last state table and maintain it in the conjunction to the bitemporal table. We would then define a key for the new table (in the example above Employee Number), and keep only the latest information there. The inserts to that table could be done in parallel to the bitemporal table or, depending on the business needs, this table could be rebuilt periodically from the bitemporal table.

A third solution could be archiving any older data on a different disk altogether. That way the access to the newer data (and the latest state) would still be fast enough, and the queries that need older data would still be possible.

In addition, the space requirement increases to accommodate extra time stamp information. If our table is relatively narrow then the overhead of the time column could be significant. Let s revisit our quoting example and assume the original table looks something like this:

10:01:12.010	37.82	37.83	400	300

The size of each record in this table would be 4+4+4+4+4 = 20 bytes. If we add valid start time and valid end time the size of each record would become 4+4+4+4 +4+4+4= 28 bytes. The difference of 8 bytes is a 40% increase in storage and can translate to a 2.5 gigabyte increase in storage requirement per day for NYSE quotes. For high speed analytics that require all of the data to be in memory this number is quite significant.

There is no cookie-cutter solution for this difficulty and every business problem should come with its own considerations. For instance, in the quoting example we know that the new quote always supersedes the previous one. Therefore valid end times are implicit and need not be inserted into the database. In addition, we may need to make an assumption that unless that quote is superseded, the current quote in the system is the valid quote.

4 Kdb+ as a Solution to Bitemporality

There are a number of ingredients that need to be introduced by a database vendor to become competitive in time series space.

It should be somewhat apparent by this time that businesses need to incorporate bitemporality to grow and prosper. The question becomes what is the best way to do that. Traditional relational database management systems are not very well equipped to deal with this problem, because they ignore the important fact that the data is ordered by time. All of the updates come in sequential time series order and, therefore, are best stored and operated on in vector form. Therefore, a columnar-based database is essential in adding bitemporality and serious time-series capabilities.

There are a number of other ingredients that need to be introduced by a database vendor to become competitive in time series space. One is the ability to run parallel operations on the data. This requires a technique to divide an operation on large data into several operations on smaller sub-tables and then to aggregate the solutions. In addition, the ability to easily manipulate times and dates within the database would have significance with the implementation of any solution.

There are plenty of technologies that exist to help with this problem, including OneTick™ database, Vertica and many others. Out of the abundance of products, one in particular stands out and that is kdb+ database. Kdb+ was developed by Kx Systems and comes with its own proprietary query and scripting language Q. The company has been in business for over 15 years and has pioneered a column-oriented database design. Because all of the data in a kdb+ database is organized in columns, it is extremely efficient to retrieve and manipulate individual columns as vectors. Further, because kdb allows operations that depend on the order of a table, it is well suited to moving time series queries.

In addition, kdb+ provides a multitude of attributes and physical database layout options that can be applied for optimal storage and faster query executions.

There is another important advantage of a kdb+ database and that is the Q programming language that comes bundled with it. There are many mathematical functions built in already and they are optimized to work on vectors. Of particular interest are the functions that can utilize map-reduce techniques. Map reduce is extremely important concept when analyzing large quantity of data because it allows the program to parallelize some calculations.

Finally, one feature useful feature of a kdb+ database is the fact that all times, and timestamps (to the nanosecond) are represented as basic data types, making time-ordered analysis extremely fast. This, coupled with a number of different joins available, makes it very easy for a Q programmer to manipulate and store time columns in a natural and consistent way.

Kx has developed and continues to update all of the needed ingredients for time-series analytics. Kdb+ includes different data storage and indexing options to allow easy use map-reduction techniques. This in turn allows a massive parallelization of queries, which is a number one requirement for a time-series database.

Kdb+ provides more functionality than discussed in this paper. For full details please visit **http://kx.com.** If you are looking for a way to add bitemporality to your business, what kdb+ provides out of the box is, in most cases, all you need to optimize your business model and improve your bottom line.

About the Author

Jeffrey Shmain is a Senior Manager leading the Time Series practice within the Business Intelligent Management group in Capgemini's Financial Services Global Business Unit. He has over ten years of experience in the financial industry delivering business critical software solutions. Prior to joining Capgemini, Jeff worked for Banc of America Securities and NYSE Euronext.

The author would like to thank **Dr. Dennis Shasha** of New York University and **Simon Garland**, Chief Strategist for Kx Systems for their assistance with this paper.

For more information, please visit **www.capgemini.com/financialservices** or e-mail **financialservices@capgemini.com.**