// WHITE PAPER

zizo

Business Intelligence Tools & Big Data:
Why Linear Scalability Is No Longer Enough

Purpose

This paper considers the interoperation between the **ZiZo**® Pattern Database and Business Intelligence (BI) tools, with particular focus on the performance characteristics of the database that make it particularly well suited to this application. The paper contrasts this with the alternative, which is storing the same data set in a Relational Database Management System. In doing this, the paper emphasises how **ZiZo**® differs from the majority of in-memory databases which are broadly in-memory implementations of the relational model. **ZiZo**® is something different; it is a pattern database that has been written specifically with the idea of running in memory, which also happens to be able to run SQL.

This makes it ideally suited to acting as the data store for the large volumes of data that are going to be queried using a BI tool. Another unique aspect of **ZiZo**® is that the data store is not static. It is constantly evolving in response to the way it is being used. This is similar to the way in which the human brain reacts to external stimuli to bring relevant memories to the fore and lets older less critical items drift into the background to be recalled only when needed. Also, like the brain **ZiZo**® constantly forges links between patterns, retaining the ones that have utility.

Throughout, this paper will use a couple of sample SQL statements as motivating examples. These statements were created by a real BI tool in response to user actions. These queries have been chosen because they help to highlight some of the issues while at the same time are far from atypical as BI tools frequently produce SQL requests that the experienced eye will see are difficult to run efficiently against large data sets. This is natural and reminds us that if a BI tool is to be employed against large data sets, then the performance that you see can be critically dependent on the quality of the SQL that the BI tool will generate.

These two queries are:

```
SELECT CAST(MAX(CASE WHEN field==NULL THEN 1
WHEN NOT(field==NULL) THEN 0 ELSE NULL END)
AS TINYINT) AS Column_1_1 FROM table;
```

and:

```
SELECT CAST(MAX(CAST(CASE WHEN field > 0 THEN 1
WHEN NOT (field > 0) THEN 0 ELSE NULL END AS INTEGER))
AS TINYINT) AS Column_1_1 FROM table;
```

At first sight these appear to be innocent requests which, while not perhaps as elegant as a "hand-crafted" request, certainly don't correspond to the complex and verbose SQL that is frequently generated by BI tools. It is worth remembering when our analysis is complete that if these two simple queries can have so great an impact on performance, then how much worse a truly typical BI query could be without **ZiZo**® to help out.

zizo

Business Intelligence Tools & Big Data:
Why Linear Scalability Is No Longer Enough

Business Intelligence (BI) Tools & Big Data

BI tools are playing an increasing role in the data estates of many big businesses. They hold out the promise of offering the end user the opportunity to do self-service reporting and analysis on data. This can make the business more responsive and also free up expensive DBA time. However, this freedom comes at a cost. BI tools are by their nature unconstrained, allowing the user to ask almost any question of the data. This places an increased strain on the underlying database because this kind of behaviour is very difficult to optimise for.

However let's return to the earlier example and recall that these queries were issued by the BI tool simply in response to opening a data source, and will therefore occur no matter what the user does or plans to do. Let us further assume that for one million rows this query takes one second to run and determine whether or not a null value is present. This is probably a gross over-estimate, but as we shall see, it really doesn't matter. When we scale up to one billion rows the same query is going to take 15 minutes to run and as there is one of those queries for every field in the table, we can see how at this scale it might take hours just to establish a connection to such a data source.

Some databases can optimise this query and recognise that it can stop as soon as it finds a single null value. This will help to reduce the overall time, but if nulls are rare, or don't actually exist in that column, it is still going to have to run a full table scan. Another approach is to parallelise the query. It is certainly true that a query like this can be made to run a lot faster by distributing the work. However, a significant number of cores would be needed to restore a one second response time for this query, which would require a very big investment in hardware. The reason for all of this expense is very simple; relational database technology was never designed to operate at this scale of data.

Linear Scalability is no longer good enough

When relational database technology began to emerge forty years ago it was based on set algebra, and its performance characteristics for complex queries were essentially **O(n.log(n))**. This was fine for small tables with thousands of rows, but it was soon swamped by data growth. New relational database technologies sprang up. These used indices and aggregates to accelerate queries and meant these databases would scale linearly as data grew. However, there was a cost. You cannot compute or store every possible index or aggregate. They had to be chosen in advance of delivery based on a study of how the database was going to be used. If this was wrong, or new requirements emerged, then the whole design could be compromised and performance would soon become unacceptable. Of course if our requirement is to connect to a BI tool, we may be beaten before we even start because the behaviour of users of these tools is so difficult to predict.

All the major vendors are now offering in-memory versions of their products. However, these are still, for all their reliance on columnar stores etc. in-memory versions of relational databases whose performance scales linearly with data size. Put enough data against them and they will either run out of time or run out of memory.

zizo

Business Intelligence Tools & Big Data:
Why Linear Scalability Is No Longer Enough

Try a living pattern database instead…

Compare this to how **ZiZo**® works. **ZiZo**® doesn't store data in rows or columns, but instead considers a row to be a point in a living space of patterns. Patterns can be structured as simple sets of possible values for fields or more complex patterns that are combinations of other patterns, or groups of patterns that are linked by a common property. All of the data exists in the pattern space, but it is no longer held in a fixed structure. When it is time to query the data zizo® does not directly run the query it was given, instead it transforms the query into an equivalent property of the pattern space and then tests if that property holds. For example the query

```
SELECT CAST(MAX(CASE WHEN field==NULL THEN 1
WHEN NOT(field==NULL) THEN 0 ELSE NULL END)
AS TINYINT) AS Column_1_1 FROM table;
is transformed to the pseudo SQL
SELECT CAST(HAS_NULL(field) AS TINYINT) AS Column_1_1 FROM table;
```

The engine must then recover the pattern set associated with the field and test if the set contains a null value. The time required to locate the pattern set is O(log(p)) where p is the number of pattern sets in the pattern space, and the time to search the set for a value is O(log(m)) where m is the size of the pattern set itself. Although both p and m can be very large numbers it is reasonable to assume for a large n (the number of rows in the data) that both p and m will be less than n, so we can assume a bound on the time to run this query as O(log(n)). To help to establish the general applicability of this approach let us look at the second query.

```
SELECT CAST(MAX(CAST(CASE WHEN field > 0 THEN 1
WHEN NOT (field > 0) THEN 0 ELSE NULL END AS INTEGER))
AS TINYINT) AS Column_1_1 FROM table;
This can be rewritten as
SELECT CAST((MIN(field)>0) AS TINYINT) AS Column_1_1 FROM table;
```

ZiZo® can execute this by recovering the pattern set for field again and identifying the smallest value in the set. Again both of these steps are bounded by O(log(n)). Clearly log(n) performance is greatly to be preferred over linear performance when the data size is large.

So what is the catch? Just like aggregates and indices there are an infinite number of possible pattern sets, so which ones should be in memory? At this point we see one of the beauties of **ZiZo**®: its pattern space is not static, but is live and evolving. This pattern space is initially seeded with a carefully chosen set of patterns. These patterns require time O(n.log(n)) to compute, but this price is only paid once, up front, when the raw data is loaded. These "seed patterns" have the highly desirable property that based on these pattern sets, any other pattern set can be computed in linear time. The **ZiZo**® database starts from these seed patterns and proceeds to populate the pattern space in response to the requests that come in from users, predicting future needs based on past behaviour. There are two clear concerns that stem from this; firstly that the predictions will not be correct, and secondly that in so doing **ZiZo**® will quickly fill the available memory. If we start with the worst scenario

zizo

Business Intelligence Tools & Big Data:
Why Linear Scalability Is No Longer Enough

which is that, the pattern set is not present at the time that the query needs it, meaning that it must be generated. This will take time O(n), so we are no worse off than with an RDBMS anyway. However, it turns out that with a large user base **ZiZo**® quickly acquires sufficient samples to be 90% or more accurate in its predictions on large data sets. As the users focus in on detail its accuracy drops, but in turn so does the cost of computing the pattern set because n is smaller. It is also the case that **ZiZo**® may get its predictions wrong and compute sets that are not needed, but as we shall see this has little impact on performance.

Every pattern set that is put into the pattern space is tagged with two values which are the age of the set and the cost to compute it. Both are measured in milliseconds. The age actually measures how long it is since the set was last accessed, so it will be reset to 0 each time a user employs it to answer a query. When the pattern space begins to become full a cull of the space will take place which removes the oldest and cheapest pattern sets first. This minimises the impact of the cull on user response times. Further, if **ZiZo**® makes a wrong prediction, then the set will be added to the space, but the cull process will naturally notice that the set has aged and so it will inevitably be culled and then cease to have a further impact on performance.

This approach works because the time to solve a query in the presence of the required pattern sets is so short (usually fractions of a millisecond) that there are plenty of spare cores and cycles for the process of maintaining the pattern space. For example, in a major retailer **ZiZo**® supports 4000 users querying 8 billion rows of sales transaction data with sub-second response times yet the whole solution runs on hardware costing less than $30,000.

Conclusions

Big data presents an unprecedented challenge to the world of unconstrained Business Intelligence queries. Not only is it the case that queries that scale linearly are no longer sufficient, but also the triggering of these queries is outside of the control of the IT team that maintains the system.

The performance of BI tools is constrained by the performance and particularly the ability of the underlying database to optimise these queries. Although many BI tools come with their own data stores that are optimised for the tool, these stores do not scale to sizes larger than the 10s of millions of rows. Similarly there are often native connectors between the BI tool and the underlying database that enables the tool to use that database as a replacement for its own data store. However, this represents a hidden lock-in associated with the tool preventing switching database because of the impact that has on performance. Furthermore, many BI companies have sizeable revenue streams that derive from the production of these native connectors so these companies are not incentivised to produce an “open” SQL layer that can be run efficiently. The simple existence of a SQL interface to a BI tool is no guarantee that the tool can scale through that interface.

The example queries have shown us that linear scalability is no longer good enough in the face of predicted data growths. To use these tools efficiently you need a database whose response times are better than linear. Such performance is not possible if the database itself is static because of the impossibility of determining what values will be required prior to the user interacting with the BI tool. Therefore a database like **ZiZo**®, that is agile and is continually re-organising its pattern space, just like your brain manages its memory, offers a means to deploy the best of modern BI tools against the huge data sets that we are going to encounter in the future.