WHITE PAPER

zizo

Data Is Changing Everything. So Are We.

THE BENEFITS OF A PATTERN DATABASE

BY TOM LONGSHAW

This paper describes what a pattern database is, what advantages a pattern database has over conventional database technology and how zizo's implementation of a pattern database exploits these differences and improves on them.

Contents

WHAT IS A PATTERN DATABASE?

A pattern database is a distinct class of data representation separate from either row or column based storage for data. In fact, a pattern database does not directly store the data at all. Instead it stores patterns from which the data can be recovered if needed.

To best understand what a pattern database is you need to first understand the process by which it is built. The builder is presented with the source data one row at a time. It treats each row is a source of patterns that it "learns" and stores. When it finds a pattern that has already been learned, then it will not store it second time, but just adds a pointer to the first instance. It also keeps and stores a running total of the number of times it has seen a pattern. The next question to answer is "What is a pattern?". There is no single answer to this, but the commonest answer is that a pattern is an instance of a value for a field in the table. Another class of pattern is a compound pattern, which is two or more patterns that are combined to create a pattern. Now by combining patterns together repeatedly you can create a root pattern that represents one complete record. Thus the builder converts the data into a collection of root records, one per row and stores these as the representation for the data.

Having understood the process, we can move on to look at a generalised model for the representation. There are three key parts to this model:

Metadata

The Metadata contains information about the nature of the data held; such as the name of the field, its type and also navigation data that tells you how to find the field in the pattern space.

Index

The index is a top-level index into the pattern space. Each entry in the index identifies a pattern in the pattern space which is a root pattern that represents a complete record.

The Pattern Space

The pattern space is just collection of patterns. They can be stored in any order and some patterns will contain references to other patterns stored elsewhere in the space.

We can now determine that this representation is a lossless compression of the original data. It is a compression because the patterns are de-duplicated, meaning as we add more data the pattern space grows more slowly than the data size; ergo the more data you put into a pattern database the more efficient it becomes. The representation is lossless because you can use it to recover the original data. This is achieved by starting from the index and for each entry in the index you recover the root pattern and then recursively expand that pattern until you have only primitive patterns left. In this way you have recovered the contents of an entire record.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk

Before concluding this section let us briefly touch on the subject of information. Formally the information content of a set of data is a measure of the minimum number of bits the must be stored in order to losslessly recover the original data. We can immediately see that we are doing something similar in creating a pattern database. In a pattern database we are not seeking to minimise the storage cost but rather to balance the storage and retrieval costs to create an efficient database. However, the transformation that we have done enables us to operate in the "information space" rather than the "data space". This has important theoretical implications that we will come to later.

DISADVANTAGES OF A PATTERN DATABASE

Most of the rest this document will talk about the advantages of a pattern database. However it's only fair to discuss some of the disadvantages too.

Relational databases have one enormous benefit. They are designed from the bottom-up to solve queries by reading data in a strictly linear fashion. The vast majority of SQL queries can be resolved in this way, which means that they can make efficient use of data stored on disk and also make extensive use of pre-fetch and pipelining found on modern CPU's.

A second benefit of an RDBMS is that when it comes to adding data into a table the process is quick and efficient. In contrast a pattern database has quite a complex build process. It also needs a lot of memory to store and navigate the pattern space during the build. We shall see how the build can be speeded up when we discuss zizo's implementation of a pattern database, zizo®.

However, there is no getting away from the non-linear memory access required to implement a pattern space. A typical pattern will be composed from several other patterns that are "scattered" around the pattern space. Algorithms for clustering patterns to try to achieve linear access are defeated by the de-duplication which means that gains from one clustering are offset when the same pattern is paired with a different pattern elsewhere in the data.

This means that if the pattern space is to operate efficiently it needs to be held in random access memory (RAM) not on disk. This has typically constrained the way in which a pattern database can be used. In particular if the pattern space for a table is larger than the available RAM there will be queries and operations that cannot be performed.

One final limitation that is worth being aware of when implementing a pattern database is that even modern operating systems such as Windows, Linux and O/SX have memory management restrictions the mean that the largest contiguous block of memory you can work with is 4 GB. This can present problems when a set of patterns is going to be larger than this limit.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk

GENERAL BENEFITS OF A PATTERN DATABASE

The first and most obvious of these is that a pattern database makes much more efficient use of the RAM than an RDBMS. Its de-duplication strategy means that it can represent and work with many more records in RAM than even a modern RDBMS implementation that makes heavy use of row or column caching. However, it should be noted that as the amount of available RAM and disk performance improves this benefit becomes more marginal.

The real benefit of a pattern database comes from its ability to operate in the information space rather than the data space. Put simply when a pattern database runs a query it will typically transform the query into something that can be evaluated directly in the pattern space. This avoids any work in reconstituting the original data, but also makes the general performance much more efficient. Let us consider some simple examples to begin with. Consider a query like ...

SELECT COUNT(*) FROM table WHERE manufacturer='Ford'

The query engine can use the pattern space to determine the pattern identify for 'Ford'. This is typically an integer. If it can't find an identifier for 'Ford' then it knows that it never learned that pattern, so there can be no records that match and the answer to the query is 0. Assuming 'Ford' is present, then we can retrieve the pattern for 'Ford' and read how many times it appears in the data and that is answer. Note that this query has been answered in the way that is completely independent of the number of records in the table!

Let us consider a slight variation on this query which is ...

SELECT COUNT(*) FROM table WHERE year_of_manufacture < 2000

We will assume that '2000' is in the pattern space. In zizo's implementation of a pattern database we have added some extra rules to the pattern space that will assist us. One of these is the rule that the pattern space respects the natural ordering of the data. This means that for two patterns A and B in the same field of the table if

B > A —› id(B) > id(A)

Where id (A) is the pattern identifier for A. We can use this to identify precisely the patterns that will satisfy

year_of_manufacture<2000

and then just sum their frequencies. Note that the time to perform this query is proportional to the size of the pattern set for year_of_manufacture not the number of records.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk

Our next example combines the last two queries. We want to do . . .

SELECT COUNT(*) FROM table WHERE manufacturer='Ford' AND year_of_manufacture < 1990

It's now unlikely that we built a pattern that satisfies this condition. If we assume that the IDs we retrieve for 'Ford' and '1990'are x and y respectively then we must iterate over the root index retrieving the pattern IDs for manufacturer and year_of_manufacture such that they are equal to x and less than y. The query performance is now proportional to the number of records. However, we're still operating in the information space this means that:

- We don't need to retrieve the actual pattern values from the records
- All comparisons are between integers which are very fast operations on modern CPUs
- In the zizo implementation we have structured the pattern space in a way such that the operation to retrieve the pattern ID from the root accesses memory in a sequential fashion allowing us the benefits of pipelining and pre-fetch

There is one final example of a selection that we can consider. This example is particularly problematic for a conventional RDBMS

SELECT COUNT(*) FROM table WHERE manufacturer IS LIKE 'F%%d'

This kind of query will typically trigger what is known as a full table scan where every record has to be read from disk.

We can again process this in the information space. Our first action is to process the pattern set for 'manufacturer' and identify the IDs for the patterns that match the wild card 'F%%d'. Now using these IDs we can process the root index counting the elements that point to a pattern with one of these identifiers.

So far many of the operations we have talked about could be achieved by using a well written tokenising scheme. Even then many databases dispense with partial ordering of the tokens in order to improve update performance. However, the next query demonstrates a capability that is unique to a pattern database. Let us consider the query . . .

SELECT COUNT(*) FROM table GROUP BY manufacturer

This is essentially asking the pattern database to produce a new index where record access is in order of manufacturer. As we shall see in a pattern database the time to do this is still directly proportional to the number of records.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@zizo.co.uk

www.zizo.co.uk

We begin with the observation that each pattern in manufacturer we know the number of instances of that pattern and the total number of records. Let us say that we have 1000 records and four manufacturers (the last column in the table is the number of occurrences).

Audi	1	85
BMW	2	138
Ford	3	412
VW	4	365

We create a new empty index with a thousand spaces in it and an array with four values in it (0, 85, 223, 635) (running totals from the frequencies). We now proceed as follows. We take the first item in the index and retrieve the pattern ID for manufacturer for that item, say it is three (Ford). We now get item three from our array and place the index one to a position 223 in the empty index. Finally we increment item three in the array so we now have (0, 85, 224, 635). Remove from entry two in the index which has pattern ID 0 so we can place it a position zero in the new index and the array becomes (1, 85, 224, 635). We continue like this for the remaining 998 index entries. At the end of this time we will have filled all the slots in the array and grouped the original data by manufacturer.

Let us just review what we just did. We sorted the data set by a field in time proportional to the number of records. We could have done this for any field in the data. Now one of the fundamental results in computer science is that the time to sort a set of data is 0(n log n); that is proportional to n.log(n) where n is the number of records in the data. This is true when you operate in the data space, but not in the information space. In the information space the data can be reordered in time 0(n).

Once we know how to sort data in linear time it turns out that we can perform a lot of other traditionally hard database operations in linear time.

We can sort by multiple fields simply by reversing the order of the fields we wish to sort by. We sort by the last field first, and then pass the index from this sort into the sort by the next to last field and so on until all the fields used. At this point the index will be sorted by all the requested fields.

We can perform a “distinct “ by sorting on the requested field(s) and then just take the first record in each group.

We can even perform the join between two tables in linear time. To do this we sort both tables by the field(s) on which they are to be joined and then perform a pair-wise traversal of the two lists joining those where the keys match. Note that the result of the join is just another set of patterns that are added to the pattern space. These patterns just point to the pairs of patterns representing the root records that are joined.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk

INTRODUCING ZIZO®

ZIZO'S IMPLEMENTATION OF PATTERN DATABASE SOLUTIONS

zizo® improves on the basic implementation we have just described. It does this mainly through improvements in the way in which memory is used and managed in the database.

As we said earlier one of the biggest issues with using a pattern database is that for it to work efficiently the pattern space needs to be held in RAM. Although the cost of RAM has dropped greatly over the past decade data requirements have kept growing. Despite using pattern technologies it still isn't possible to operate a database with billions of rows in it on an affordable platform using a pure pattern database.

The first problem we tackled was the RAM requirements during the build phase. Building a pattern database has always required more RAM than operating the same database requires. This is because at the build stage we do not know the limits to the pattern space, so less memory efficient structures like hash tables and binary trees have to be used to model and index the pattern space while it is "being grown". To address this we segment the source data in various ways into subsets that can be processed individually to form a part of the pattern space. Each of these sub-spaces is written to a separate file and at the end we have an assemble process that combines the contents of the individual files into the representation for the entire pattern space. This keeps the memory requirements manageable. It has a second benefit which is that using one machine where there is ample memory allows us to use separate threads to construct each sub-space. By using this parallelism we can provide a data load process that is both quick and efficient in the memory it uses.

zizo® addresses the problems of RAM usage during query by using what we call paged memory solutions. Rather than operate solely in memory our pattern database mixes memory and disk access to expand the capacity of a pattern database. We do this in two ways; first making better use of the memory and second when there is no more RAM available being intelligent about how the memory is recycled.

To make better use the memory we have coded up pattern space in a way that is inherently "sparse". We only load into memory the parts the pattern space that are required to answer the questions being asked. We use a "demand mechanism" to infer what is required and markers in the pattern space to show what is already populated. When a query hits a region that is unpopulated the patterns are brought in from disk but any parts of the table that have not been queried remain only on secondary store.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk

Secondly we make as much use as we can of what we have in memory by using smart caching. When queries result in the computing of derived subsets of the data these sets are cached. This means that a subsequent query can often be resolved without reference to the original patterns but instead directly from these caches. This could mean that some patterns can be removed from RAM without preventing the queries functioning.

Thirdly when RAM does start to run low we are intelligent about what we page out. As every entry in the pattern space and cache has a timestamp associated with it we know when these patterns were last read. If memory needs to be freed, then we can make sure that it is the least used patterns that are released and by using a reference counting mechanism we also ensure that the patterns are only released when no one is still using them. This avoids the need for slow and complex locking mechanisms; and the expiry mechanism even provides a safe and natural way to deal with updates to tables. When a table changes the new requests will only access patterns from the new instance meaning no new queries will be touching the older patterns, and as such they will naturally be cleared out of memory once all queries referring to them have finished.

There is one further way in which zizo® and their clients benefit from utilising a pattern database; a reduction in design activities as a result of using this technology. Performance in a pattern database is dictated by the volume of information in the underlying data rather than the data itself. The information content exists independent of the representation you choose which means that in the first instance we can leave our automated systems to import and build the data with no intervention from us. We are confident that the resulting tables will be ready to query and will perform well. However, we cannot say our processes end up being design free. As result of our interaction with the data we may conclude that further patterns derived from the original data would help to improve performance or enhance the user's understanding of the data. If this is the case, we will often extend the pattern space with these derived patterns.

However, in the end we know that because the pattern database holds all the information that was originally present in the data and it will deliver queries in a scalable fashion our will find engagement with zizo® to be both timely and rewarding.

For further information
please call us
01908 366662
or 07894 762766

email: matthew.napleton@
zizo.co.uk

www.zizo.co.uk