A Technical Report "Off the Record": Using Alternative Data Models to increase Data Density in Data Warehouse Environments.

Victor Gonzalez-Castro Lachlan MacKinnon
School of Mathematical and Computer Sciences,
Heriot-Watt University.
Edinburgh, U.K. EH14 4AS
victor@macs.hw.ac.uk lachlan@macs.hw.ac.uk

Abstract

In Data Warehouse environments data sparsity is a common issue. In the past few years a number of different techniques have been proposed to tackle this issue, but it remains an unresolved problem. The majority of the techniques developed to date to tackle this problem have been based on the relational model. At the same time other research has focused on alternative data models that abandon the traditional record storage/manipulation structure. We are investigating the use of alternative data models (Triple Store, Binary Relational, Associative, Transrelational™) to increase the internal data density in Data Warehouses, with the intention of reducing the data sparsity and thereby decreasing the final size of the database

1. Introduction

This work brings together commercial experience and action research in the Data Warehouse industry with both historical and current database and data model research.

Experience of building and maintaining data warehouses for a number of different companies has consistently demonstrated that the databases that constitute the warehouse grow at an exponential rate, with the result that in a few months huge database sizes are reached. It is not uncommon to talk of Terabyte-size tables inside Data Warehouses.

There are a number of reasons for this:

1. Depth of History maintained
2. Level of data granularity
3. Number of dimensions involved
4. Dimension levels
5. Precalculation of all possible aggregate level query answers
6. Data Completeness
7. Data sparsity
8. Ability of the Data Base Management System to handle data sparsity
9. DBMS capacity to handle and recover space in fragmented table spaces
10. DBMS data compression features (if any)

The phenomenon of combining the dimensions and precalculating all possible aggregate levels is known as *Database Explosion,* and the original data set can grow by factors of tens or even hundreds [6].

The reasons previously listed have been tackled in different ways, for example:

1. *Depth of History maintained*. It depends on the business need to maintain historical data, but what we have found is that frequently organizations limit the number of years to be maintained in the Data Warehouse in order to deal with the storage and processing limitations. By doing this they lose a significant level of the richness of a Data Warehouse.
2. *Level of data granularity*. It is not uncommon to follow the rule to keep 15 months of data at the lowest level (a complete year and a quarter) to be able to do the traditional *this-year-last-year analysis*, and then start consolidating data at a higher level of granularity..
3. *Number of dimensions involved*. Remember that the practical use (or meaning) of dimensions is to represent the different points

of view of the organization's data. It is a common practice to limit the points of view by limiting the Data Warehouse design to a particular business area (Finance, Marketing, Human Resources, etc.) and thus, to a certain extent, create Data Marts instead of a real Enterprise Data Warehouse.

4. *Dimension levels*. One common decision made during the conceptual design of a Data Warehouse is to limit the number of dimension levels (Hierarchy depth) to the most common level of analysis.
5. *Precalculation of all possible aggregate level query answers*. This is one of the most difficult points while building a Data warehouse, the developers need to balance the desired online answer time and the batch calculation time, in order to complete all the possible combinations required. This trade-off is considered and decisions are made on which possible aggregated queries will be pre-computed. Some OLAP products make suggestions on what are the most common queries to be considered to precalculate, and which will be calculated "on the fly".
6. *Data Completeness*. It should not came as a surprise that during the conceptual design of a Data Warehouse the end user wants some analyses that they regard as interesting, but while reviewing the existing source systems (OLTPs) the basic data required to build these analyses does not exist. While this simply means that the user has to forget about these interesting analyses, the reality is that the user may well regard these as paramount and considerable additional data may have to be generated to mitigate this situation. Another common problem with data completeness is that in Legacy systems some fields may have been used in a different way from the original design (i.e. A field to keep the address is used to keep previous employee position)
7. *Data sparsity*. There is almost nothing that can be done with data sparsity, because if there are "holes" (nulls) in the data (i.e. data does not exist for a particular cell) we cannot create any information to fill that hole. Especially in Relational (and particularly in SQL Language) the treatment of nulls has been a significant area of research, but it has been more oriented to OLTP systems and has not been properly solved. See [7] chapter 8.
8. *Ability of the Database Management System to handle data sparsity*. Commercial RDBMS (i.e. Oracle, SQL Server, DB/2, etc.) are built on SQL capacities and limitations [11]. This issue is discussed in more detail later in this paper, but it is clear that in practice commercial RDBMSs do not handle sparsity well.
9. *DBMS capacity to handle and recover space in fragmented table spaces*. This is handled in different ways by the RDBMS but those that have a deficient space recovery mechanism are not well suited for Data Warehouse environments. The main reason for this is that if the RDBMS requires to unload and reload the table to recover the space this is not a practicable solution for Terabyte-sized tables, because of the temporary (swap) space required for the operation as well as the processing time
10. *DBMS data compression features (if any)*. RDBMS avoid the use of traditional compression methods such a Huffman coding and Lempel-Ziv to reduce the size of the Database. This is principally because of the CPU cost involved in undertaking compression and decompression operations. [14].

Reasons 1 to 5 are to some extent manageable by the design of the Data Warehouse schema, and it is not uncommon to find that developers have a number of tricks to deal with these problems and also the limitations of their products. Thus they avoid demonstrating any of those limitations by ensuring that their products operate within their safety zone.

Reason 6 depends to a greater extent on the Data Quality of the OLTP source systems and the Extraction, Transformation and Loading processes (ETL).

However, reasons 7 to 10 are out of the developers control and are more dependent on the internal technology used by the DBMS.

Our current work is focused on reason 8 and we will explain more in section 3.

2. Background

Data Warehousing usually involves the use of Multidimensional modeling of data. Multidimensional modeling is a data representation that arranges the data

to take into account the different business views of that data.

Those different points of view of the data are called *Dimensions*. The data can be organized in different groupings or consolidation levels in each dimension, and inside each level there are the actual member values or specific instances of the levels (for example in the time dimension at the Month Level we have January, February, March, … , December).

Another important element of Multidimensional modeling is called *The Facts,* these are the things that the organization wants to analyze (for example, the value of sales).

Multidimensional models can be stored in Relational Databases (RDBMS) or in Multidimensional Databases (MDBS). Multidimensional modeling allows us to view and analyze the data by using techniques called On Line Analytical Processing (OLAP). When OLAP techniques are used against Relational Databases, this is called Relational On Line Analytical Processing (ROLAP), and when it is done against a Multidimensional Database, it is called Multidimensional On Line Analytical Processing (MOLAP).

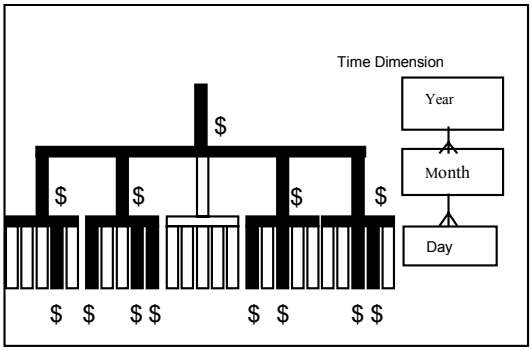

Fig.1. A three level dimension and nulls. After [6].

One of the main characteristics and advantages of OLAP is the ability for an end-user to easily manipulate data. So the end user can query any possible combination of the data existing in the Data Warehouse by combining different aggregation levels, for example "give the total daily sales by product for the last three days". (See Fig. 1). As can be seen it is necessary to calculate the total sales of all products on a daily basis (Cartesian Product). And of course not all products were sold in the last three days, so many products can have Null sales on any day.

This example illustrates how holes can exist in data. As identified earlier, this is called Data Sparsity and it is related to the data density.

Different approaches had been taken to answer the user queries, some precalculate all the different combinations and store them while others do not precalculate them and carry out the calculation on the fly, depending on the end user query.

Figure 2 is a graphical representation of a Cartesian product of two dimensions and shows how Data sparsity is originated and its relation with data density.

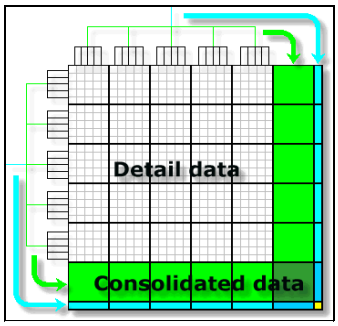

Fig. 2. Data sparsity and density, from [6]

3. The problem

Reason 8: *Ability of the Database Management System to handle data sparsity*.
This problem is of considerable interest to us, especially since all RDBMS offer poor management of sparsity (missing information). Codd in [7] made a fundamental analysis of missing information on Relational/SQL systems and he suggested the need for ***a fundamental change to the relational model Version 2, the use of a 4 value-logic*** to deal with missing information. To the best of our knowledge, at this time, ***No one has implemented this fundamental change***.

Codd accepted that implementing the 4 valued-logic described in the Relational Model Version 2 (RM/V2) was not enough and he identified the future work necessary for Relational Model Version 3. Citing Codd's words as they appear in [7] chapter 9.

"I do not claim that RM/V2 handles the "half-missing" case, in which a specific and precise value is unknown, but either a small range of possible values is known, or else there is a high probability that the missing value is one of a very few values. This case may not be ignored in RM/V3, but it is now more urgent for DBMS

products to handle the cases for which RM/V2 provides support. In any event, it will be necessary to show that the extra machinery (hardware and/or software) need to support the "half-missing" case is going to pay its way."

The relational Model Version 2 was originally developed in 1979 [12] and was consolidated in 1990 [7]. In the many years since then no commercial RDBMS has implemented Codd's suggestion to handle sparsity. Unfortunately, Codd never completed Version 3 of the Relational Model.

It is, however, clear from this that Relational technology inherently suffers from a deficiency to deal with data sparsity, especially the SQL Language. Another Codd citation from [7]:

"Overall, the SQL approach to handling missing values is quite disorganized and weak. This will lead to disorganized thinking on the part of users, an increased burden for them to bear, and many unnecessary errors that are not discovered can lead to incorrect business decisions based on incorrectly extracted data."

Given these fundamental weaknesses in the relational model, we are researching on the use of alternative data models to build Data Warehouses, with a view to determining whether any of these approaches will decrease data sparsity. However, before we consider these we should address some existing approaches in existing commercial models.

From the practitioner's point of view some approaches to deal with data sparsity have been addressed with multidimensional structures, although the issue remains unresolved on the fundamental model.

Hypercubes: are a single-cube logical structure, this definition applies to multidimensional databases or Relational OLAPs (in this case is a single Fact Table).

Multicubes: on this approach the application designer segments the database into a set of multidimensional structures each of which is a subset of the overall number of dimensions in the database. This approach is used by products like Express, Pilot, Holos, TM1 and Microsoft Olap Services [13]. Exponents of these systems emphasize their great versatility and potentially greater efficiency (particularly with sparse data). ROLAP products can also be multicubes if they can handle multiple base fact tables, examples of these are Microstrategy, Informix and CA.

Still from the practitioner's position there are some techniques to avoid the database explosion [6]:

- Avoid fully pre-calculating any multidimensional object with more than five sparse dimensions to keep the Compound Growth Factor (CFG) around 2. CFG is explained below.
- Reduce the sparsity of individual data objects by good application design and by using a multicube approach, so the object has the minimum number of necessary dimensions. This has the effect of making the data denser, which reduces the CFG and reduces the number of compounding events, so Data Warehouses can shrink considerably.

CFG is a measurement defined in [6] as the square root of 5.83 per dimension involved in the Data Warehouse structure. Figure 3 presents the relation of CGF against the data density.

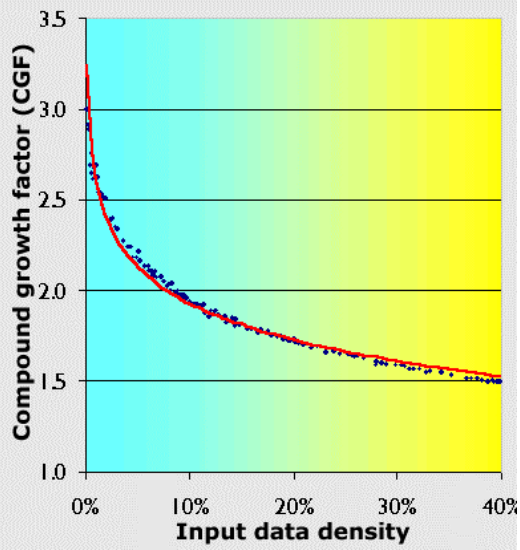

Fig. 3 Compound Growth Factor vs. Data Density. From [6].

While it is hard to predict the exact value in advance, it is reasonable to assume a CGF of about 2.0. Which is moderately sparse data. This means that adding an extra dimension with no increase in the amount of input data will at least double the size of a fully computed database. If there are more than 5 dimensions, the CGF will grow by 0.1 for each extra dimension. [6]

Using the idea of making data denser, and having seen Fig. 3, we have the following statements to research:

1. Data Sparsity is directly related with Data Density
2. Data density in Data Warehouses can be improved by using alternative Data models.
3. If we increase the Data Density, the data sparsity will be reduced.
4. The final size of the Data Warehouse will be reduced.

4. State Of the Art

Relational DBMS write the physical records as the user thinks or will use. There is a research interest in storing the data in different internal structures by abandoning the classical "Record" as the storing/processing unit and thinking of the data. There are a number of different approaches to this vision.

We intend to compare several alternative Data Models:

- Relational Model [7].Uses tuples as storage/manipulation structure (loosely speaking, they are mapped to records). It is the base model against which the other models will be compared.

- The Transrelational Model ™ [5]. It *keeps the Relational* model itself *but abandon the record storage structure*.

- The Triple store [1] [2]. This model does not use a record structure; instead it uses a structure called *the name store* to keep all the names, *and triplets* to construct the processing structure.

- The Binary Model. [4]. It considers that all tables in the system are *binary tables*.

- The Associative Model [3]. It comprises two types of data structures *Items* and *Links*. It differs from binary and Triple store in one fundamental way; Associations themselves may be either the source or the target of other associations. It uses *Quadruplets* instead of *Triplets*.

5. Comparison of Data Models

In this section we review the basic storage/manipulation structures used by each data model. Details of how each data model or product builds or uses such structures are beyond the scope of this paper, references that provide such information are given in each data model.

5.1 Relational.

Codd's seminal paper [8] defined the basic of Relational model. It uses Relations and Tuples as structures. They are materialized as Tables and Rows in commercial products. Tuples correspond (loosely speaking) to file records. Primary Keys are the identifiers for tuples.

	1	2	3	4	5
	P#	PNAME	COLOR	WEIGHT	CITY
1	P1	Nut	Red	12.0	London
2	P2	Bolt	Green	17.0	Paris
3	P3	Screw	Blue	17.0	Oslo
4	P4	Screw	Red	14.0	London
5	P5	Cam	Blue	12.0	Paris
6	P6	Cog	Red	19.0	London

Fig. 4. Relational Table. From [5]

5.2 TransRelational ™.

Invented by Steve Tarin, [9]. It does not try to replace the Relational model, instead it works at a lower level, closer to the physical storage

The idea of Transrelational is to separate the values and the linkage information that ties those values together. In a direct image system (Record structure) both parts are together because the linkage information is represented by physical contiguity while in Transrelational they are kept separate.

It has two basic structures:

- **The Field Values Table**. Each column of the table contains the values from the corresponding field of the file, rearranged into *ascending sort order* and without duplicate values in each column. (Fig 5)
- **The Record Reconstruction Table**. Its cells are not supplier numbers or supplier names (etc.) any longer, instead they are row numbers. It is used to reconstruct the record. (Fig 6)

	1	2	3	4	5
	P#	PNAME	COLOR	WEIGHT	CITY
1	P1	Bolt	Blue	12.0	London
2	P2	Cam	Green	14.0	Oslo
3	P3	Cog	Red	17.0	Paris
4	P4	Nut		19.0	
5	P5	Screw			
6	P6				

Fig. 5. Field Values Table of Figure 3. Form [5]

	1	2	3	4	5
	P#	PNAME	COLOR	WEIGHT	CITY
1	4	3	2	1	1
2	1	1	4	6	4
3	5	6	5	2	6
4	6	4	1	4	3
5	2	2	3	5	2
6	3	5	6	3	5

Fig. 6. Record Reconstruction Table for Fig.3.

Date in [5] explains how these structures are built and how they work together.

5.3 The Triple Store.

In Sharman and Winterbottom's seminal paper [1] they define the Triple Store structure consisting of two basic storage structures:

- **The Name Store**. It stores all the information related with the "real world", called names. It is a structure with two columns one for the names and other for the unique identifiers which are generated automatically and are unique. (see Fig.7).
- **The Triple Store**. It stores the triplets that are ordered 3-tuples, <i1, i2,i3>; Where each element of the triple is an identifier which must have been generated by the name store before the triple is created.(see Fig. 8).

i11	i12	i13
i21	i22	i23
…	…	…
in1	in2	in3

Fig. 7. The Name Store.

	Name
i1	name 1
i2	name 2
…	…
i4	name n

Fig. 8. The Triple Store.

As can be seen all the "real world" information is stored in one single repository (the Name Store), for all "columns", there is no differentiation in the information stored. The repository must be able to store these names as lexical objects. The Triple Store keeps the structure of the names (how to rebuild the record).

5.4 The Binary Model.

Chen's 1976 paper "The Entity-Relationship Model – Towards a Unified view of Data"[10] is widely credited as the origin of the binary model, although there was considerable earlier work. IBM's Laboratory at Hursley Park, England, has maintained an interest in storage structures for the Binary model that dates from P J Titman's 1974 paper "An experimental Database Using Binary relations" through to Sharman and Winterbottom's 1988 paper "The Universal Triple Machine: a Reduced Instruction Set Repository Manager"[1]

The basic idea is that all the tables in the system are Binary tables with one surrogate key plus an attribute.

To rebuild the record it is necessary to execute *joins* between the binary tables.

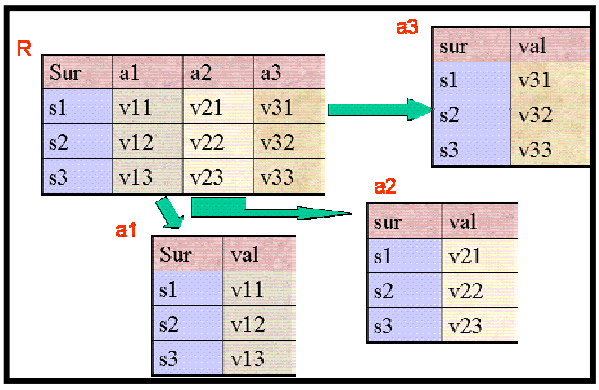

Fig. 9. The binary model Table decomposition

5.5 The Associative Model.

Published by Simon Williams in [3], databases that use the Associative Model, store all different types of data in one single structure (as TripleStore) while relational databases use unique structured tables to store each different type of data.

The Associative Model of data uses two types of data structures.

- **Items**, each of which has a unique identifier, a name and a type. (Fig 10)
- **Links**, each of which has a unique identifier, together with the unique identifiers (this is the difference from Triple store) of three other things, that represent the source, verb and target (this is the Triple store). (Fig 11)

As can be seen again the *Items* structure (Fig.10) keeps all the data and the *Links* are used to reconstruct the record.

Identifier	Name
77	**Flight BA1234**
08	**Heathrow Airport**
32	**12-Aug-98**
48	**10:25 am**
12	**Arrived at**
67	**On**
09	**At**

Fig. 10. Items in the Associative model.

Identifier	**Source**	**Verb**	Target
74	77	12	**08**
03	74	67	**32**
64	**03**	**09**	**48**

Fig. 11. Links in the Associative model.

The associative model differs conceptually from the binary model and in implementation from triple stores in one fundamental way: Associations themselves may be either the source or the target of other associations.

It uses *Quadruplets* instead of Triplets and by this, according with Williams [3], it adds more flexibility.

6. Contribution

Having carried out the initial industrial and commercial analysis, and related this to the existing research models, we now move in to the next phase of our work. The aim of the next phase of our research is to carry out an impartial survey and comparison of whether or not the use of alternative data models can improve Data Density in Data Warehouse environments and at the same time observe the effect that such data density increase has on Data Sparsity.

As seen the chosen models abandon the idea of a Record structure, instead they store all the alphanumeric data in one repository (each one uses their own name and data structure) and the means to reorganize the information or reconstruct the Record in another structure (each one for each data model).

All of them try to store each value just once in a repository (Values Table, Name Store, Items, Binary Table) and then use this value wherever required by relating it to many records via the reconstruction structure (Record reconstruction table, Triplets, Links, Binary Association table). See Table 1.

Model	Storage Structure	Linkage Structure
Relational	Table (Relation)	By position
Transrelational	Values Table	Record Reconstruction table
Triple Store	Name Store	Triple Store
Binary	Binary Table	Joins
Associative	Items	Links

Table1. Comparison of data model structures

By avoiding duplicate values (at field level and of course at Record level), a reduction on the final database size is expected. To prove this, we intend to use a Database implementation of each data model:

- Relational Model – A commercial RDBMS.
- Triple Store Model – TriStarp.
- Binary Model – Monet DB.
- Associative Model – Sentences DB.
- Transrelational – since there is currently no instantiation of the Transrelational model available, we will build an implementation of the essential algorithms.

We will use the TPC-H data set with the Data models/products mentioned above, to run a set of benchmark metrics to determine relative performance, and then consider relative data density and sparsity.

This work builds on work already undertaken in action research in industry, and reported briefly earlier, but also seeks to bring together several longstanding areas of research to address a current unresolved commercial problem. Given the huge growth in Data Warehousing in recent years, the future of the relational model as the data model of choice may well be under threat.

7. References

1. Sharman G.C.H. and Winterbottom N. The Universal Triple Machine: a Reduced Instruction Set Repository Manager. Proceedings of BNCOD 6, pp 189-214, 1988.
2. TriStarp Web Site: http://www.dcs.bbk.ac.uk/~tristarp .Updated November, 2000.

3. Williams Simon, The Associative Model of Data. Second Edition, Lazy Software Ltd. ISBN: 1-903453-01-1 www.lazysoft.com
4. MonetDB. ©1994-2004 by CWI. http://monetdb.cwi.nl
5. Date, C.J. An introduction to Database Systems. Appendix A. The Transrelational Model , Eighth Edition. Addison Wesley. 2004. USA. ISBN: 0-321-18956-6.
6. Pendse Nigel. Database explosion. http://www.olapreport.com Updated Aug, 2003.
7. Codd, E.F. The Relational Model for database Management Version 2. Addison Wesley Publishing Company. ISBN 0-201-14192-2. 1990. USA.
8. Codd E.F. A relational model of data for large data banks. IBM Research Laboratory, San Jose, California. Communications of the ACM. Volume 13, Number 6, June 1970.
9. U.S. Patent and Trademark Office: Value-Instance-connectivity Computer-Implemented Database. U.S. Patent No. 6,009,432 (December 28, 1988).
10. Chen, Peter Pin-Shan. The Entity-Relationship Model – toward a unified view of data. Massachusetts Institute of Technology. ACM transactions on Database systems, March 1976. Volume 1 Number 1.
11. C.J. Date, Where SQL Falls short, Datamation May 1, 1987
12. Codd. E.F. "Extending the Database Relational Model to Capture More Meaning". ACM Trans on Database Systems 4:4.
13. Pendse, Nigel. The OLAP Report. Multidimensional data Structures. www.olapreport.com . Last updated March 19, 2001.
14. O'Connel S.J., Winterbottom N. Performing joins without decompression in a compressed Database system. SIGMOD Record, Vol. 32, No. 1, March 2003.