An Experimental Consideration of the Use of the Transrelational™ Model for Data Warehousing*

Victor Gonzalez-Castro[1], Lachlan M. MacKinnon[2], and David H. Marwick[1]

[1] School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, EH14 4AS, Scotland
{victor, dhm}@macs.hw.ac.uk
[2] School of Computing and Creative Technologies, University of Abertay Dundee, Dundee, DD1 1HG, Scotland
l.mackinnon@abertay.ac.uk

Abstract. In recent years there has been a growing interest in the research community in the utilisation of alternative data models that abandon the relational record storage and manipulation structure. The authors have already reported experimental considerations of the behavior of Relational, Binary Relational and Associative models within the context of Data Warehousing, to address issues of storage efficiency and combinatorial explosion through data repetition. In this paper we present an implementation of the Transrelational™ model, based on the public domain definition provided by C.J. Date, which we believe to be the first reported instantiation of the model. Following the presentation of the implementation, we also present the results of performance tests utilising a set of metrics for Data Warehouse environments, which are compared against a traditional N-ary Relational implementation. The experiment is based on the standard and widely-accepted TPC-H data set.

1 Introduction

The Transrelational™ model was defined by S. Tarin and patented in the United States of America [14], and it has been promoted to the Database community by C.J. Date through a series of seminars and in the latest edition of his widely-adopted textbook [3]. However, as far as we can determine there is no implementation available for either commercial or research use. Therefore, in order to carry out our experimental consideration, we have utilised the general description made by Date [3] of the Transrelational™ model and its behavior to implement the essential algorithms that make up the model. Since Date [3] has provided the only public domain documentation of the model, which we shall henceforward refer to as TR following his nomenclature, we shall make reference extensively to his work in describing our experiment.

Our experimental consideration of the TR model follows on from research which we have already reported considering the performance of Relational, Binary Relational and Associative models in the context of Data Warehousing [4][5][8]. We

* The Transrelational model is based on the Tarin Transform Method and is the intellectual property of Required Technologies Inc.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 47 – 58, 2006.

are interested in the use of alternative data models that can solve the problems of the inefficiency of storage caused by models based on rows, which include repetitions at field level no matter if they use normalized (snow-flake) or non-normalized (star) schemas, as well as the database explosion phenomenon [7] that occurs in Relational Data Warehouses.

According to Date [3:954], the TR model has an implicit data compression mechanism by using *condensed columns* which eliminates the duplicate values at a column level. This is very appealing in Data Warehouse environments where it is common to have many repetitive values at column level, so we wanted to measure and benchmark this characteristic of the model.

2 The Transrelational Model™

As already indicated the only public domain description of the Transrelational™ model is provided by Date [3]. In this section we introduce some of the basic definitions of the model in order to establish a baseline for our experimental consideration, all quotes and references to Date are attributable to [3] and where necessary page number references are supplied. As a starting point Date states that, *"The Transrelational™ (TR) is not intended as a replacement for the Relational model"*, from which we can infer that it can be seen as an alternative route to implement the relational model and thus to build a Relational DBMS.

From the early days of data processing systems, through the development of relational databases and up to the present day, data has predominantly been conceived as Records of *n* number of fields or attributes. This approach has been called an *N-ary Storage Model* (NSM) [2] or in Date's nomenclature *Direct Image Systems* (DIS). Within this approach data is seen (and stored) following a horizontal approach (rows).

Alternatively, there is also a vertical approach (columns) to store and process data, and this has its origins in Copeland's seminal paper *"A Decomposition Storage Model"* (DSM) [2]. This has recently been used as the basis for the creation of some novel database architectures and instantiations, such as MonetDB [1], [6], SybaseIQ [11], [12] and C-store [10]. TR differs from both vertical and horizontal approaches, but is closer to a vertical approach since, in the *Field Values Table* (FVT), each column stores distinct values, and most of the processing can be done at column level. This is analysed in more detail in section 3.

2.1 Model Data Structures

To illustrate the characteristics of the model, we utilise the examples developed by Date [3]. The TR model consists basically of two storage structures: the **Field Values Table (FVT),** where each column contains the values from the corresponding field of the input file rearranged in an ascending sort order **Fig. 1(b);** and the **Record Reconstruction Table (RRT) Fig. 1(c),** which keeps a set of values derived from the original input file that can be thought of as pointers to enable the original record to be rebuilt when necessary using the **ZigZag algorithm** [3:948].

Record Sequence	S#	SNAME	STATUS	CITY
1	S4	Clark	20	London
2	S5	Adams	30	Athens
3	S2	Jones	10	Paris
4	S1	Smith	20	London
5	S3	Blake	30	Paris

	S#	SNAME	STATUS	CITY
1	S1	Adams	10	Athens
2	S2	Blake	20	London
3	S3	Clark	20	London
4	S4	Jones	30	Paris
5	S5	Smith	30	Paris

	S#	SNAME	STATUS	CITY
1	5	4	4	5
2	4	5	2	4
3	2	2	3	1
4	3	1	1	2
5	1	3	5	3

Fig. 1. (a) A Suppliers relation, (b) Field Values Table, (c) Record Reconst. Table

To understand how both tables are used when rebuilding a record, utilising the ZigZag algorithm, we provide Date's description:

"Step 1: Go to cell [1, 1] of the Field Values Table and fetch the value stored there: namely, the supplier number S1. That value is the first field value (that is. the S# field value) within a certain supplier record in the suppliers file.

Step 2: Go to the same cell (that is, cell [1, 1]) of the Record Reconstruction Table and fetch the value stored there: namely, the row number 5. That row number is interpreted to mean that the next field value (which is to say, the second or SNAME value) within the supplier record whose S# field value is S1 is to be found in the SNAME position of the fifth row of the Field Values Table -in other words, in cell (5,2) of the Field Values Table. Go to that cell and fetch the value stored there (supplier name Smith).

Step 3: Go to the corresponding Record Reconstruction Table cell [5, 2] and fetch the row number stored there (3). The next (third or STATUS) field value within the supplier record we are reconstructing is in the STATUS position in the third row of the Field Values Table-in other words, in cell [3,3]. Go to that cell and fetch the value stored there (status 20).

Step 4: Go to the corresponding Record Reconstruction Table cell [3, 3] and fetch the value stored there (which is 3 again). The next (fourth or CITY) field value within the supplier record we are reconstructing is in the CITY position in the third row of the Field Values Table-in other words, in cell [3,4]. Go to that cell and fetch the value stored there (city name London).

Step 5: Go to the corresponding Record Reconstruction Table cell [3, 4] and fetch the value stored there (1). Now, the "next" field value within the supplier record we are reconstructing looks like it ought to be the fifth such value; however, supplier records have only four fields, so that "fifth" wraps around to become the first. Thus, the "next" (first or S#) field value within the supplier record we are reconstructing is in the S# position in the first row of the Field Values Table-in other words, in cell [1,1]. But that is where we came in, and the process stops."

To this point, the model provides no potential database size reduction because all values are kept and additional data is held in the Record Reconstruction Table, we refer to this as TR Version 1. The desired reduction is achieved when the **Condensed columns** are introduced. As can be observed in Fig. 1(b) a considerable amount of redundant data is stored, this is also true in traditional N-ary systems (see Fig. 1(a)). The Column-Condensing process aims to eliminate such redundancy by keeping unique values at column level; we refer to this as TR Version 2.1. This process should

	S#	SNAME	STATUS	CITY
1	S1	Adams	10 [1:1]	Athens [1:1]
2	S2	Blake	20 [2:3]	London [2:3]
3	S3	Clark	30 [4:5]	Paris [4:5]
4	S4	Jones		
5	S5	Smith		

Fig. 2. Condensed Field Values Table with row ranges

be applied selectively, since attempting to condense columns already consisting of unique values does not make sense. However, as each value can be used in many "records" it is necessary to keep **Row Ranges** (numbers in squared brackets in Fig. 2) to avoid losing information on how to reconstruct the record, we refer to this as TR Version 2.3. The resulting FVT with condensed columns and row ranges is presented in **Fig. 2.**

The Column-Condensing process destroys the unary relationship between the cells of the FVT and the cells in the RRT, but the ZigZag algorithm is easily adaptable as stated in [3:956].

"Consider cell [i, j] of the Record Reconstruction Table. Instead of going to cell [i,j] of the Field Values Table, go to cell [i',j] of that table where cell [i',j] is that unique cell within column j of that table that contains a row range that includes row i."

3 Implementation

We have implemented algorithms to create the Field Values Table, the Record Reconstruction Table and the Zigzag algorithm to rebuild the records. Some variations and improvements have been made during implementation and we will describe those in the following subsections.

This implementation was focused on the initial bulk load of the Data Warehouse and it retained the limitations identified by Date [3:943] where updates are discarded and the Database is Read Only. Inherently, Data Warehouse environments are more suited to these assumptions (with batch updates during off line hours and read only during operation hours) than transactional systems. Consequently, we would argue that the benchmarking of the TR model for Data Warehouse environments is both relevant and important.

An extraction tool was written in order to generate the flat files that will be processed by the TR model algorithms. One important point that was introduced during the extraction process is that each record is pre-appended with its record number, to provide additional support during the Transformation and Loading stages.

3.1 The Field Values Table

Data within each column is rearranged into ascending sort order. All operations are made in bulk and parallelising as much as possible, extensive use of the sort,

parallelisation and synchronisation mechanisms offered by the operating system has been made. The sorting process of each column is made in parallel as each column is independent. Improvements were introduced in order to prepare the creation of the RRT and minimise reprocessing. The first step is to create a structure (we call it Working File WKF and it enhances the algorithms described by Date) where each column is in ascending order and maintains the original record number as sub-index Fig. 3(a). From this structure (WKF) the Field Values Table with condensed columns (Fig. 2) is derived by choosing unique values in ascending order and recording the corresponding row ranges. The other structure derived from the WKF structure is the Permutation Table [3:951] Fig. 3(b), which is required to build the Record Reconstruction Table.

The Column-Condensing process is selective as recommended by Date [3:953]. In our implementation we establish that columns with unique values and those where the reduction would be lower than the established condensation threshold are not condensed. The condensation threshold is not part of Date's algorithms and we define it as the ratio between the original number of values (including repetitions) and the number of unique values. The condensation threshold is 30% and this was set after testing different threshold values and considering the balance between the disk space required to keep row ranges versus the disk space required to keep the complete column with relatively few repetitive values. The processing time was also considered to set the threshold. This approach was taken in order to maintain a balance between the theoretical model and the pragmatic implementation.

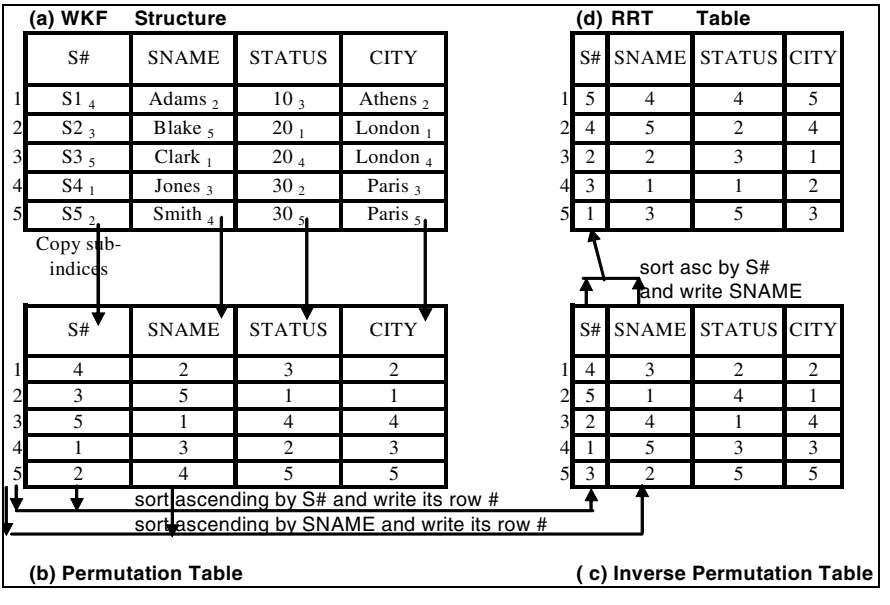

Fig. 3. Proposed alternative generation algorithm to build the Record Reconstruction Table

3.2 The Record Reconstruction Table

A permutation is defined by Date [3:951] as the ordering of the records by a particular column. For example the permutation corresponding to ascending S# ordering is 4,3,5,1,2 (refer to Fig. 1(a)). According to this definition, the *Permutation Table* is computed directly from the original input file, but instead of doing it in this way we derived it from the WKF structure by using the already ascending ordered values but taking the sub-indices, see Fig. 3(a) and 3(b). Thus the columns of the Permutation Table can be computed in parallel. The first column of the Permutation Table keeps an ascending sequence from 1 to the number of records in the table (i.e. the row number within the Permutation Table).

Date also defined the *Inverse Permutation as, "That unique permutation that if applied to the original sequence 4,3,5,1,2, will produce the sequence 1,2,3,4,5".* It is computed column by column on the permutation table by applying the previous rule to obtain the ascending sequence 1,2,3,4,5.

We found that is more efficient to compute the Inverse Permutation Table from the existing Permutation Table by taking each column together with its corresponding row number and then sorting ascending by the corresponding column values and writing the row number in the Inverse Permutation Table rather than the column values (in this example S#), see Fig 3(b) and 3(c). The resulting Inverse Permutation Table is exactly the same as that described by Date, and its columns can be computed in parallel.

Finally the Record Reconstruction Table can be built from the Inverse Permutation Table. Date's algorithm [3:952] is as follows:

"Go to the cell [i,1] of the Inverse Permutation Table. Let that cell contain the value r; also the next cell to the right, cell [i,2], contain the value r'. Go to the rth row of the Record Reconstruction Table and place the value r' in cell [r,1]."

Following the algorithm for i=1,2,3,4,5 will produce the entire S# column of the Record Reconstruction Table. The other columns are computed in a similar way.

This algorithm processes one row at a time which is inefficient. We propose the following algorithm to compute the Record Reconstruction Table:

From the Inverse Permutation Table use column i and column i+1, sort in ascending order by i-th column values and write the value held in i+1 column to the corresponding i-column in the RRT. This applies to all columns except the last one which must use the n-th column and the 1st column. See Fig. 3(c) and (d).

Our algorithm enables bulk processing and each column is processed in parallel as there is no dependency between columns.

3.3 Implemented Versions

Four versions of the TR model were implemented in order to gather information regarding its behavior with different characteristics. Each version, its characteristics, and the implications for the ZigZag algorithm [3:948] are listed in **Table 1.**

Table 1. TR model versions

Version 1	Field Values Table (FVT) keeps repetitive values. Both the Record Reconstruction Table (RRT) and the ZigZag Algorithm are as stated in [3].
Version 2.1	All columns in all tables are condensed in their corresponding FVT tables and the remaining unique values keep row ranges. The ZigZag Algorithm is enhanced to be aware of row ranges. The RRT Table remains unchanged.
Version 2.2	Only the Fact Table is considered to condense its columns. The ZigZag algorithm needs to be improved to detect the Fact Table and be aware that row ranges only exist in the Fact Table but not in any other table. RRT remains unchanged.
Version 2.3	Selective column condensation in the FVT is applied if the established threshold is reached; this is applied to all tables as proposed by Date. The ZigZag algorithm needs to be aware of condensed and uncondensed columns, and those which are condensed have row ranges. RRT Table remains unchanged.

4 Benchmarking Environment

In order to benchmark the TR model and its implementation in Data Warehouse environments the standard and well accepted TPC-H [13] data set was chosen. TPC-H has the characteristics of real life data warehouses where a big Fact table exists with complementary tables around this table (no matter if it is a Star or Snow-flake schema). The authors are very experienced in the use of this data set for benchmarking Data Warehouses considering different data models [4],[5],[8] to highlight their specific characteristics. The tests were executed with two database sizes, called scale factors (SF=100 MB and SF=1GB).

The machine used to evaluate the defined metrics has 1 CPU Pentium IV @ 1.60GHz, 512 MB in RAM, Cache size 256 KB, Bus speed 100MHz and Operating System Fedora 2 version 2.4.9-12. The relational instantiation used is Oracle Version 9.0 with its corresponding SQL*Plus and SQL*Loader utilities.

SHQL version 1.3 [9] is used to provide a SQL interface. It was used to execute the necessary DDL statements. The implemented algorithms made use of the initial structures generated by SHQL to build and manipulate the required tables (FVTs and RRTs).

5 Experimental Results and Analysis

The results presented in this section follow the flow of the Extraction Transformation and Loading Process. As mentioned before, a tool was made to extract data and generate the input flat files to be loaded in both Relational and TR instantiations. The differences between input flat file sizes are small but the TR input files are slightly bigger because of the extra column required to keep the row number that will be used for further processing. Resulting flat file sizes are presented in **Table 2.**

Table 2. Extracted file sizes to be used as input

Scale Factor (SF)	Relational (MB)	TR (MB)
100 MB	102.81	103.77
1GB	1,049.60	1067.46

The next step is to Transform and Load the input files into the instantiations for both models. As stated before four different versions of TR were implemented (Table 1). The results for these versions are presented in **Table 3.** (SF=100MB) and **Table 4.** (SF=1GB).

Table 3. DBMS Object size in MB with SF=100MB

TPC-H Table Name	Relational	**TR V1** (with repetitive values)	**TR V2.1** (everything condensed)	**TR V 2.2** (only Fact Table condensed)	**TR V2.3** (selective condensation)
Region	0.0625	0.0005	0.0005	0.0005	0.0005
Nation	0.0625	0.0024	0.0027	0.0024	0.0023
Supplier	0.1875	0.1600	0.1900	0.1600	0.1600
Customer	3.00	3.04	3.41	3.04	2.79
Part	3.00	3.41	2.34	3.41	2.12
PartSupplier	13.00	14.19	13.90	14.19	13.05
Orders	19.00	25.63	21.91	25.63	18.56
Lineitem	88.00	137.07	86.09	86.09	86.90
TOTAL	**126.31**	**183.50**	**127.84**	**132.52**	**123.58**

Table 4. DBMS Object size in MB with SF=1GB

TPC-H Table Name	Relational	**TR V1** (with repetitive values)	**TR V2.1** (everything condensed)	**TR V 2.2** (only Fact Table condensed)	**TR V2.3** (selective condensation)
Region	0.0625	0.0005	0.0005	0.0005	0.0005
Nation	0.0625	0.0024	0.0027	0.0024	0.0023
Supplier	2.0	1.75	2.07	1.75	1.68
Customer	27.0	32.00	36.23	32.00	29.3
Part	30.0	36.37	24.74	36.37	21.52
PartSupplier	128.0	149.00	137.69	149.00	130.93
Orders	192.0	274.96	235.32	274.96	200.89
Lineitem	848.0	1489.71	925.73	925.73	962.22
TOTAL	**1,227.1**	**1,983.78**	**1,361.81**	**1,419.80**	**1,346.54**

TR Version 1 is of limited value because it keeps all repetitive values and adds more information within the RRT table; as a result it increases the N-ary Relational Instantiation DB size by a factor of around 50%.

From Tables 3 and 4 with version 2.1 (where all columns are condensed); in medium sized tables (Supplier and Customer) the effect of condensing resulted in bigger table sizes than the corresponding sizes without being condensed, this is as a result of keeping the row ranges; for the bigger tables (Lineitem and Orders) the Column-Condensing process is beneficial. Considering that the Lineitem Table (Fact Table) is the biggest table, Version 2.2 was introduced. The benefit of this version is that only the Fact Table is passed through the Column-Condensing process in order to reduce the CPU processing time, but the downside is that the final DB size is bigger than Version 2.1. Finally in version 2.3 all tables are processed by the Column-Condensing process but in a selective fashion where if the estimated condensing level reaches the established threshold then the column is condensed, otherwise there is no benefit in investing CPU processing time which will not achieve significant column size reductions. According to the results obtained, version 2.3 is the one that offers a better balance between processing time and disk space consumption but requires a complex ZigZag algorithm to rebuild records when necessary. The ZigZag algorithm needs to be intelligent enough to identify condensed columns and uncondensed columns and make use of row ranges when rebuilding the original record.

These analyses have been focused on the FVT since, as identified by Date [3:954] the FVT will offer compression benefits. However, our experimental results show that, even when condensing repetitive values, the final database size is bigger or, at best, only slightly smaller than the traditional N-ary Relational instantiation.

Further analyses based on Version 2.3, show that the FVT behaves in keeping with Date's description, but the problem is with the RRT. While the RRT at first sight only holds integers, after millions of rows (i.e. Lineitem=6 million rows for SF=1GB), these integers are of more considerable size and occupy most of the final database space. These results are presented in **Table 5.** (SF=100MB) and **Table 6.** (SF=1GB).

As in any Data Warehouse environment the Fact table (in this case LineItem) occupies most of the space, with SF=1GB this table occupies 962MB of 1,346MB of the total DB size. Importantly, however, the corresponding RRT for Lineitem required 760MB of those 962MB. In general RRT structures are 65% of the total DB space while FVT structures are the remaining 35%. From these results it is clear that further work is necessary to tackle the RRT structures, and particularly the RRT for Fact Tables, to enable the TR model to achieve the benefits predicted.

Another key finding of the experiment is that with the bigger scale factor, the Version 2.3 (selective condensation) has better results than any other version (see Table 4), including Version 2.1 where every column is condensed, but remains very close (additional 10%) to the traditional N-ary Relational implementation.

Finally, another aspect to be considered is the time to Transform and Load the input files into the DBMS. In this aspect the TR instantiation required more time than the N-ary Relational instantiation. The Transformation and Loading time was not linear with respect to DB size, as presented in **Table 7.** with around 4 times more with SF=100MB and 10 times more with SF=1GB.

Table 5. Transrelational™ RRTs and FVTs with SF=100MB

TPC-H Table Name	Relational (MB)	TR V2.3 (MB)	RRT (MB)	FVT (MB)	% RRT vs total	%FVT vs total
Region	0.0625	0.0005	0.0001	0.0004	11%	89%
Nation	0.0625	0.0023	0.0003	0.0020	13%	87%
Supplier	0.1875	0.1600	0.0300	0.1300	19%	81%
Customer	3.00	2.79	0.68	2.11	24%	76%
Part	3.00	2.12	1.04	1.08	49%	51%
PartSupplier	13.00	13.05	2.68	10.37	21%	79%
Orders	19.00	18.56	8.95	9.61	48%	52%
Lineitem	88.00	86.90	66.36	20.54	76%	24%
TOTAL	126.31	123.58	**79.74**	**43.84**	**65%**	**35%**

Table 6. Transrelational™ RRTs and FVTs with SF=1GB

TPC-H Table Name	Relational (MB)	TR V2.3 (MB)	RRT (MB)	FVT (MB)	% RRT vs total	%FVT vs total
Region	0.0625	0.0005	0.00005	0.00045	10%	90%
Nation	0.06	0.0023	0.00030	0.00200	13%	87%
Supplier	2.0	1.68	0.37	1.31	22%	78%
Customer	27.0	29.3	8.06	21.24	28%	72%
Part	30.0	21.52	12.29	9.23	57%	43%
PartSupplier	128.0	130.93	31.41	99.52	24%	76%
Orders	192.0	200.89	51.69	149.20	26%	74%
Lineitem	848.0	962.22	760.34	201.88	79%	21%
TOTAL	1,227.1	1,346.54	**864.16**	**482.38**	**64%**	**36%**

Table 7. Transformation and Loading Times

	Scale Factor (SF)	Transformation & Loading time
Relational	100MB	2.4 minutes
Transrelational™	100MB	10.2 minutes
Relational	1GB	19.9 minutes
Transrelational™	1GB	191.6 minutes

6 Conclusions and Future Work

The TR model as described by Date is very appealing for Data Warehouse environments, but after analysis of our results it really does not offer the tangible

benefits that we were looking for. It may reduce the inefficiency of storing repetitive values at column level that exists in traditional N-ary Relational implementations, but the expected reduction in Data Warehouse size was not achieved. This has to be set against the results achieved for other alternative models [4][5][8], especially Binary Relational. However we have been able to produce a novel public domain instantiation of the TR model described by Date, and we have identified and implemented improvements to the algorithms of that model. Further research needs to be done on the RRT structure to minimise its size and thus reduce the final DB size, but at the same time we envisage that this will increase the processing time. Date [3:956] identifies another feature that could be used in TR (which we might call Version 3) which uses *Merged Columns*. However, we have identified that the existing drawbacks of the RRT should be the next problem to be tackled, and we would argue that this needs to be undertaken before introducing more complexity to the algorithms for the marginal benefits in terms of the final DB size that might be achieved in Version 3.

Based on our research results [4][5][8] and the current state of the TR model, we would argue that it does not represent the model of choice for future Data Warehouse environments.

Acknowledgments. The present work has been possible due the financial support of Consejo Nacional de Ciencia y Tecnología México (CONACYT) and Secretaría de Educación Pública (SEP) México. The authors gratefully acknowledge that support.

References

1. Boncz, Peter. Monet: A next generation DBMS Kernel for query intensive applications. PhD Thesis. Amsterdam, 2002.
2. Copeland, George P. Khoshafian, Setrag N. A Decomposition Storage Model. In Proceedings of the ACM SIGMOD Int'l. Conf. On Management of Data, pp 268-279, May 1985.
3. Date, C.J. An introduction to Database Systems. Appendix A. The Transrelational Model, Eighth Edition. Addison Wesley. 2004. USA. ISBN: 0-321-18956-6. pp.941-966
4. Gonzalez-Castro, Victor. MacKinnon, Lachlan. A Survey "Off the Record" - Using Alternative Data Models to increase Data Density in Data Warehouse Environments. Proceedings BNCOD Volume 2. Edinburgh, Scotland 2004.
5. Gonzalez-Castro, Victor. MacKinnon, Lachlan. Data Density of Alternative Data Models and its Benefits in Data Warehousing Environments. British National Conference on Data Bases, BNCOD 22 Proceedings Volume 2. pp 21-24. Sunderland, England U.K. 2005.
6. MonetDB. ©1994-2004 by CWI. http://monetdb.cwi.nl
7. Pendse, Nigel. Database explosion. http://www.olapreport.com Updated Aug, 2003.
8. Petratos, P and Michalopoulos D. (editors) Gonzalez-Castro V. and Mackinnon L.(authors). Using Alternative Data Models in the Context of Data Warehousing. 2005 International Conference in Computer Science and Information Systems. Athens Institute of Education and Research, ATINER. Greece. ISBN: 960-88672-3-1. pp 83-100. 2005.
9. SHQL. http://backpan.cpan.org/modules/dbperl/scripts/shql/

10. Stonebraker, Mike, et.al. C-Store: A column Oriented DBMS. Proceedings of the 31st VLDB conference, Trondheim, Norway, 2005. pp. 553-564.
11. SybaseIQ web site. www.sybase.com/products/informationmanagement/sybaseiq
12. Sybase Inc. Migrating from Sybase Adaptive Server Enterprise to SybaseIQ white paper. USA 2005.
13. TPC Benchmark H (Decision Support) Standard Specification Revision 2.1.0. 2002.
14. U.S. Patent and Trademark Office: Value-Instance-connectivity Computer-Implemented Database. U.S. Patent No. 6,009,432 (December 28, 1999).