A Benchmarking Technique for DBMS‘s with Advanced Data Models

Naphtali Rishe, Alexander Vaschillo, Dmitry Vasilevsky,
Artyom Shaposhnikov, Shu-Ching Chen

High-performance Database Research Center School of Computer Science
Florida International University, University Park, Miami, FL 33199
(305)348-1706, Fax (305)348-1705, {rishen,avasch01,dvasil01,shaposhn,chens}@cs.fiu.edu,
http://hpdrc.cs.fiu.edu

Abstract. The majority of database benchmarks currently in use in the industry were designed for relational databases. A different class of benchmarks became required for object oriented databases once they appeared on the market. None of the currently existing benchmarks were designed to adequately exploit the distinctive features native to the semantic databases. A new semantic benchmark is proposed which allows evaluation of the performance of the features characteristic of semantic database applications. An application used in the benchmark represents a class of problems requiring databases with sparse data, complex inheritances and many-to-many relations. Such databases can be naturally accommodated by semantic databases. A predefined implementation is not enforced allowing a designer to choose the most efficient structures available in the DBMS tested. The second part of this paper compares the performance of Sem-ODB binary semantic database vs. one of the leading relational databases. The results of the benchmark are analyzed.

1. Introduction

One of the goals of benchmarks is to compare several database management systems on the same class of applications in order to show which one is more efficient for this particular class. Benchmarks used in the industry today are always oriented towards uncovering a defined limited set of features that a database should implement efficiently in order to successfully support the class of applications the benchmark was created for.

One of the first industry benchmarks, the TP1 benchmark [1], developed by IBM purported to measure the performance of a system handling ATM transactions in a batch mode. Both TPC-C and TPC-D have gained widespread acceptance as the industry’s premier benchmarks in their respective fields (OLTP and Decision Support). TPC-E failed to garner enough support because being an enterprise benchmark, it was only relevant to a relatively small number of companies competing in that space [11]. The Simple Database Operations Benchmark [9], Altair Complex-Object Benchmark [14], OO1 [6] and OO7 [3, 2] Benchmarks were created to provide useful insight for end-users evaluating the performance of Object Oriented Database Management Systems (OODBMS).

Most object-relational DBMS are build on top of relational database by adding the following four key features [12]: inheritance, complex object support, an extensible type system, and triggers. The appearance of such databases necessitated the creation of the BUCKY benchmark [4], which tests most of these specific features. It emphasizes those features of object-relational databases that are not covered by pure relational databases.

1.1 Semantic Model

The semantic database models in general, and the Semantic Binary Model SBM [8, 10] in particular, represent information as a collection of elementary facts categorizing objects or establishing relationships of various kinds between pairs of objects. The facts in the database are of three types: facts stating that an object belongs to a category; facts stating that there is a relationship between objects; and facts relating objects to values. The relationships can be multivalued.

The objects are categorized into classes according to their common properties. These classes, called categories, need not be disjoint; that is, one object may belong to several of them. Further, an arbitrary structure of subcategories and supercategories can be defined.

1.2 Why a Different Benchmark

Unfortunately, most benchmarks do not provide general problem statement; instead they enforce a specific implementation that can not be efficiently translated to a different data model. For example, TPC benchmarks compares the efficiency of implementation of the same solution for different relational DBMS'es rather then comparing how efficiently DBMS'es are able to solve a given problem. The benchmark proposed does not enforce specific implementation thus allow native, efficient implementation for any DBMS - semantic, relational or any other, which makes it highly portable [7].

Majority of existing benchmarks is designed to evaluate features native to relational DBMS'es and none of them are suitable to evaluate performance of the features characteristic of semantic database applications. The benchmark proposed evaluates the ability of DBMS to efficiently support such features including sparse data, complex inheritances, many-to-many relations and variable field length.

The rest of the paper is organized as follows: The benchmark application and the requirements for execution of transactions are defined in general terms in sections 2, 5, 6 and 7. Sections 3 and 4 describe our implementations of this application for semantic and relational database respectively. Section 8 presents the results we obtained for a semantic and a relational database and analyses the results. Conclusions are provided in section 9.

2. The Problem Stated

It has recently become a practice for many consumer organizations to conduct consumer surveys. A large number of survey forms are mailed out to people and small companies with questions about shopping preferences in order to determine consumer patterns. Some consumers will fill out the survey and mail it back. The results of the survey should be stored in a database.

Our survey collects information about several types of legal persons. It is mailed to physical persons and corporations and we keep information about all the legal persons the survey was mailed to. Those who answer the survey and mail it back are considered consumers and categorized into the corresponding category. We also try to collect referrals from people to their friends and mail our survey to the referred people. We remember the information about referrals so that we can explore correlation between groups of people who know each other. A person may refer another person without filling out the survey and thus without becoming a consumer as we define it.

Among the other questions, we will ask a consumer which stores does he prefer to shop at. We will thus keep a catalog of stores with their names and types (pharmacy, mass merchandiser, etc.) A consumer may list several stores he usually shops at, so that many-to-many relationship is established between consumers and stores. We will also ask a consumer to tell us his approximate annual expenditures in thousands of dollars and a list of his hobbies.

We will collect information about ten types of products. Consumers will fall into different categories based on their answers about which products they regularly consume, for example "coffee drinker." We will mail them a survey where they will tell us if they regularly consume some product and answer questions about different brands they consume within each product group. Each product group in our survey has 10 brands. A consumer will tell us, which brands of the product he uses and show his preference of different brands in terms of a satisfaction rating of 1 to 4. Some consumers may indicate that they use this type of product, but none of the brands listed by us. This option should also be accommodated. We will also let a consumer write any comment about any brand he is using (from the ten brands we are asking about) if he wishes to do so. In practice, consumers seldom write any comments, so this information will be sparse. However, if they do write a comment it can be of any length and will probably be rather long.

3. A Semantic Schema for the Benchmark Database

The benchmark deals with persons and corporations. Person and corporation are represented by two different categories with the appropriate personal attributes. They both inherit from a category *Legal Person*. A category *Consumer*, will inherit from *Legal Person*, since both persons and corporations may be consumers and there is no need to distinguish between them as consumers. Since the same object can not be both a person and a corporation, categories *Person* and *Corporation* will be set up as disjoint.

A legal person who answered our survey becomes a consumer. The category *Consumer* will then be used to capture information about such legal persons as consumers. The attribute "expenditure", is thus an attribute of the category *Consumer*. The category *Consumer* is not disjoint with either *Person* or *Corporation*. In fact, every consumer must be either a person or a corporation. All of this is supported by a semantic database on the level of schema enforced integrity constraints. The relationship between categories is shown in Figure 1.

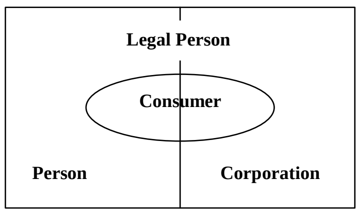

Figure 1. Relationship between subcategories

Consumers can further be categorized into inherited categories *G0, G1, ... G9*, which represent consumers of a particular product. Thus, those consumers who are soap users will be categorized into the category "Soap Users" (one of G_i). The same consumer may be categorized into several G_is if he is a user of several types of products.

In our initial database population about 50% of all objects in the database will be categorized into each of the following categories: *Legal Person, Person, Consumer* and several of the *G0, G1, ... G9* categories. Some will be categorized into

Legal Person, Corporation, Consumer and several of the *G0, G1, ... G9*. Some will be just *Person* and *Legal Person* or *Corporation* and *Legal Person*.

In addition to this, there will be some objects in the category *Store*. *Consumers* will be related to the stores they usually shop at. This relation is many-to-many, which means that a consumer may be related to several stores at once and a single store will, with high probability, be related to many consumers.

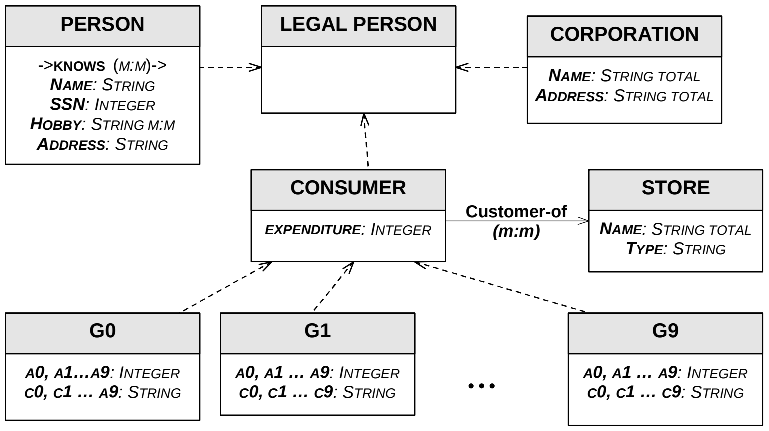

Figure 2. Semantic Schema for the Semantic Benchmark

A special relation "knows" is going from category *Person* into itself. The semantics of this is that a person may know and refer to us several other persons and will be related to each one of them via this relation. Since a person typically refers (and may be referred by) several other persons, this relation will also be many-to-many.A semantic schema of the database appears in Figure 2.

4. Relational Schemas for the Benchmark Database

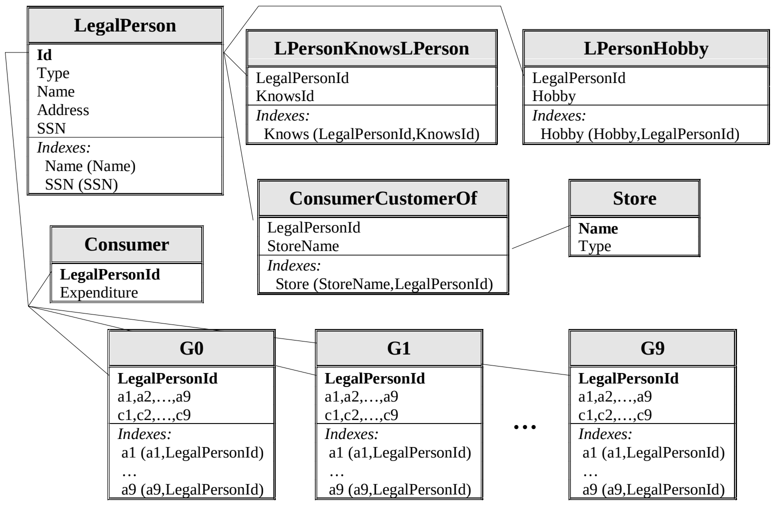

Figure 3. A relational schema for the Sparse model

While the benchmark application has a clear and obvious representation in the semantic schema, creating a relational schema for this applications in not evident. When designing the semantic schema our only concern was the readability of the schema capturing the full semantics of the application. In designing a relational schema, we have to think more about the efficiency of the queries that we expect will be executed on this database and about the tradeoffs between this efficiency, readability, and the size of the database.

Among the many choices for the relational schema we considered (for detailed analysis see [13]), we have chosen two that we believe to be the best candidates for efficient implementation. We call them the Sparse model and the Compact model. The corresponding relational schemas for these two models are shown in Figures 3 and 4, respectively. In the Sparse model the creation of ten indexes per group (consisting of LegalPersonId and a_i) is reasonable to facilitate efficient access to the data in G0…G9. Even though our benchmark queries do not access every a_i in every G_i, the schema should accommodate all similar queries with the same efficiency, so we have to create all of the indexes. However, creating so many indexes will slow down update operations and take up too much disk space and cache memory.

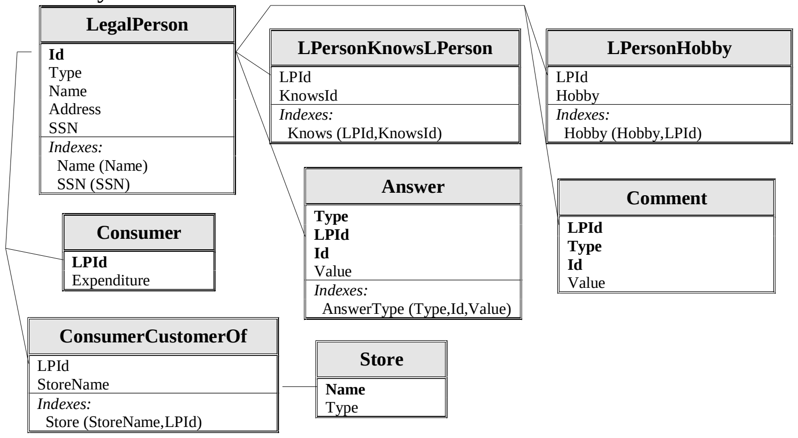

Figure 4. A relational schema for the Compact model

For the Compact model we create two tables, one to keep all a_i attributes, the other to keep all c_i attributes. Each table has the columns: LegalPersonId, group number, attribute number and the value of this attribute. The primary key will consist of LegalPersonId, group number and attribute number. For this model, we need to create just one additional index consisting of group number, attribute number and value.

	Lower bound	Upper bound	Mean	Variance
Name length	5	40	12	5
Address length	15	100	35	20
Comment length	5	255	30	100
Number of hobbies per consumer	0	19	0	10
Number of stores per consumer	1	19	4	10
Expenditure	1	89	20	10
Number of groups a consumer belongs to	1	10	5	4
Number of brands a consumer uses	0	9	1	1

Table 2. Normal distribution of parameters for initial database population

5. Initial Database Population

The number of Legal Persons in the initial population defines the scalability [7] of the database. The results published in this paper were obtained on a database with 200,000 corporations and 800,000 persons. 500,000 of persons and 100,000 of corporations are consumers. The data was generated randomly according to the table of Normal Distribution (Gaussian) parameters (Table 2). The detailed definition of the initial data set can be found in [13]. A random set of Persons must be pre-chosen for transaction #4. This set consists of 0.1% of all Persons in the database.

6. Database Transactions

Our benchmark consists of five transactions performed independently and sequentially. We expect efficient DBMS-specific implementations for each particular database.

Transaction 1:

The first transaction is simple. The task is to count the number of consumers that belong to every one of the ten product consumer groups in the database. The result is just one number: the cardinality of the intersection of groups G0...G9. The formal definition of the transaction is shown in formula (1).

$$R = \left\| \bigcap_{i=0}^{9} G_i \right\|, \qquad (1)$$

where G_i are the groups of consumers that consume a particular product.

Transaction 2:

The second transaction consists of two different physical database transactions. The first one finds all consumers who use brand #1 of product #1 as their first preference among the brands of product #1, and at the same time use brand #2 of product #2 as their second preference among the brands of product #2. Such consumers form a new group in the database G_{new}. This new group must be created in the database and all found consumers should be categorized into it. The formal definition of the transaction is shown in formula (2).

$$G_{new} = \{g \in G_1 \mid g[G_1 :: A_1] = 1\} \bigcap \{g \in G_2 \mid g[G_2 :: A_2] = 2\} \qquad (2)$$

The second physical transaction should delete the newly created group from the database. The sum of execution times of both physical transactions is considered the execution time of Benchmark Transaction #2.

Transaction 3:

The third transaction is a complex query counting those consumers who regularly shop at store X and have hobby Y, excluding those who use brand #3 of product #3 as their third preference among the brands of product #3, and at the same time use brand #4 of product #4 as their fourth preference among the brands of product #4. The result is just one number - a count of such consumers. The formal definition of the transaction is shown in formula (3).

$$R = \left\| \begin{array}{l} (\{p \in Person \mid p[Person :: hobby] = X\} \bigcap \\ \{c \in Consumer \mid c[Consumer :: customer\_of :: name] = Y\}) - \\ (\{g \in G_3 \mid g[G_3 :: A_3] = 3\} \bigcap \{g \in G_4 \mid g[G_4 :: A_4] = 4\}) \end{array} \right\| \qquad (3)$$

Transaction 4:

The fourth transaction can be explained by the following: For each person from a given (randomly chosen) set of 0.1% of all persons, expand the relation "knows" to relate this person to all people he has a chain of acquaintance to. Abort the transaction rather than commit. Print the length of the maximal chain from a person. The formal definition of the transaction is shown in formula (4).

$$\begin{gathered} NewDatabase = OldDatabase \cup K \text{ , where} \\ K = \{< s.knows.p >|\, s \in S,\, p \in Person, \exists n, \exists a_1, a_2, \ldots a_n \in Person : \\ < s.knows.a_1 >, \forall i = 1..n-1 < a_i.knows.a_{i+1} >, < a_n.knows.p >\} \end{gathered} \tag{4}$$

Transaction 5:

The fifth transaction counts the number of consumers in each one of the ten product consumer groups in the database. The result is ten numbers: the cardinality of the each of the groups G0...G9. The formal definition of the transaction is shown in formula (5).

$$R_i = \|G_i\|, i = 0..9 \tag{5}$$

7. Execution of Transactions

The benchmark is running in single user mode. Only one client is running the benchmark transactions at a time. Thus, we are not testing concurrency control performance by this benchmark. A DBMS is, however, allowed to use any kind of parallelism it can exploit in single user mode.

The benchmark transactions are executed in two modes: hot and cold. Both "cold time" and "hot time" are collected for each transaction. Both results are included in the final result. The Cold time is the time required to perform a transaction immediately after starting the DBMS on a system with an empty cache. This is normally achieved by rebooting the system before executing each transaction. The hot time is the time required to perform a transaction immediately after performing an identical transaction without clearing any cache or restarting the DBMS. To collect the hot time we run the same transaction in a loop until the time of execution stabilizes, which typically happens on the third or fourth run. Once the execution time stabilizes we compute the arithmetic mean of the following five transactions and this is considered the final hot execution time for this transaction.

8. Results and Analysis

We ran the benchmark for the Sem-ODB Semantic Object-Oriented Database Engine implemented at Florida International University's High Performance Database Research Center (HPDRC) [10]. We also ran the benchmark for one of the leading commercial relational databases. The tests were done on a dual processor Pentium II 400Mhz with 256MB total memory and 9Gb Seagate SCSI disk drive under Windows NT Server 4.0 Enterprise Edition.

The version of Sem-ODB used for benchmarking did not utilize the multiprocessor architecture of the underlying hardware. We did, however, observe the relational database using both processors for parallel computations. We have run tests with different memory limitations imposed on the DBMS'es. Sem-ODB was allowed to use 16 Megabytes of memory and never actually used more than 12 Megabytes for the benchmark transactions. For the relational database, two different tests were conducted. In one, the relational DBMS was allowed to use 16 Megabytes, in the other 128 Megabytes. For some transactions, the 16 Megabyte quota was enough for efficient execution, for other transactions the relational DBMS was willing to use up to 128 Megabytes for better performance.

Both cold and hot times were collected for each memory limitation and for both relational schemas (sparse and compact). Thus, eight execution times were collected per transaction for the relational DBMS. This was done to make sure that we have done everything we could to allow the relational database to achieve its best performance. We observed that in some cases the sparse model was more efficient, but in other cases the compact model was faster. In order to prevent criticism on the choice of the model, we decided to include all the results in this paper.

We have spent a considerable amount of time inventing different implementations and fine tuning the relational database. We tried different combinations of indexes, keys, DBMS options, and schemas in order to achieve greater performance. The semantic database on the other hand, did not require any tweaking to optimize its performance. Its performance was acceptable in the very first version.

The semantic DBMS is able to capture the exact semantics of the problem domain and provide a single optimal way to represent it in a semantic schema. All the appropriate indexes are built automatically and follow from the definition of the Semantic Database. By its design, it can efficiently answer arbitrary queries without the need for an experienced person to spend time tuning it for a particular application.

DBMS Model	**Semantic**	**Relational Sparse**		**Relational Compact**	
DB Size (Mb)	406Mb	1046Mb		382Mb	
RAM	**16Mb**	**16Mb**	**128Mb**	**16Mb**	**128Mb**
Cold times (seconds)					
Transaction 1	1.61	11.52	11.55	16.11	15.94
Transaction 2	1.13	0.53	0.56	0.34	0.36
Transaction 3	0.91	5.95	5.91	5.97	5.88
Transaction 4	55.65	55.63	43.02	55.63	43.02
Transaction 5	8.62	11.66	11.53	15.31	15.17
Hot times (seconds)					
Transaction 1	0.04	11.66	5.39	15.81	12.58
Transaction 2	0.07	0.28	0.28	0.09	0.09
Transaction 3	0.33	2.72	2.72	2.72	2.70
Transaction 4	0.23	35.02	2.87	35.02	2.87
Transaction 5	6.85	11.36	2.17	14.92	10.32

Table 3. The benchmark results

One might think that to create enough indexes for the execution of arbitrary queries, the semantic database would have to unnecessarily use too much disk space. The results however prove this not true. In our relational implementation the sparse model contains a similar number of indexes to the semantic database but requires 2.5 times more disk space. The compact model uses about the same amount of disk space as the semantic database, but has worse performance on most transactions and is not universal in the sense that this model would not be possible at all if, for example, the attributes $a_0..a_9$ were of different types.

The semantic database is outperformed by the relational on a few transactions in a cold mode. This is explained by the fact that the relational database uses a read ahead technique for reading from the disk which is not yet implemented in Sem-ODB. Table 3 shows the results for the five transactions for semantic, sparse relational and compact relational models in both cold and hot modes.

The result of the first transaction can be explained by the fact that the semantic database has a very efficient way to represent the information about the categories a particular object is categorized into, as well as the set of objects categorized into a particular category. Keeping this information is natural for the semantic database. A relational database had to gather scattered information to obtain the result. This advantage of the semantic database is especially visible in the results of the hot mode of transaction #1.

The second transaction was intended to test a simple query, a simple change in the schema of the database, and a simple update. We found out that the relational database is able to cope efficiently with this task in the cold mode, but yields to the semantic database in the hot mode. The results for the compact model are better than those for the sparse model because it does not have to change the schema and performs a simple update instead.

The third transaction is a complex query, which involves attributes of several categories in the inheritance hierarchy. As expected, the results show that the semantic database is consistently more efficient for complex queries in both cold and hot modes. The efficient inheritance implementation is the key to success for this kind of query.

The fourth transaction is a common case of pointer traversal. The results show that the cold time is much worse than the hot time since in the cold mode the database has to perform a lot of reads from random places in the physical file. In the hot time, where less reads are required and most of information is cached, the semantic database has the advantage of having an efficient representation for relations between objects, which is less efficient in the relational database.

The difference between the fifth and the first transaction is that the first transaction uses set operations to manipulate the information about categorizations of objects, while the fifth transaction only retrieves it without much processing. This information is difficult to retrieve in the compact model of the relational database. Although it is much easier to retrieve it in the sparse model, the cache size happens to be crucial here. The semantic database should be very efficient on this query, but strangely, the difference between the hot and cold times in this transaction is rather small. This means that Sem-ODB wastes too much time on some data processing after the data is already retrieved from the disk. We were surprised to see this result and will consider further improvements in Sem-ODB to eliminate this inefficiency.

9. Conclusions

In this paper, we have presented a benchmark for semantic databases. It is formulated in a way that allows efficient implementation for any type of database. We compared the ease of implementation of this benchmark for semantic and relational databases and found out that efficient implementation for a relational database requires a significant amount of fine tuning and a careful choice of indexes. The semantic implementation does not require any tuning or choosing indexes. It allows arbitrary queries to be executed. We have chosen two implementations of the relational database which were the most efficient and presented the results of both of them compared to the results of the semantic database. Although in most cases the semantic database was more efficient for this class of applications, we have identified a few inefficiencies, which need to be addressed by the database engine developers. We analyzed the results of the benchmark tests run in cold and hot mode and compared the sizes of the databases.

References

[1] Anon, et al. A measure of Transaction Processing Power. Datamation, 31(7): 112-118, April 1985.

[2] Michael J. Carey, David J. DeWitt, Chander Kant, and Jeffrey F. Naughton. A status report on the OO7 OODBMS benchmarking effort. In Proceedings of the ACM-OOPSLA Conference, pages 414-426, Portland, OR, October 1994.

[3] Michael J. Carey, David J. DeWitt, and Jeffrey F. Naughton. The OO7 Benchmark. In Proceedings of the 1993 ACM-SIGMOD Conference on the Management of Data, Washington D.C., May 1993.

[4] Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton, Mohammad Asgarian, Paul Brown, Johannes E. Gehrke, Dhaval N. Shah. The BUCKY Object-Relational Benchmark. In Proceedings of the 1997 ACM-SIGMOD International Conference on Management of Data, Tuscon, Arizona, May 1997.

[5] R.G.G. Cattell An engineering database benchmark. In: The Benchmark Handbook for Database and Transaction Processing Systems, J. Gray (ed.), Morgan-Kaufmann, San Mateo, California, 1991.

[6] R.G.G. Cattell, and J. Skeen. Object Operations Benchmark. ACM Transactions on Database Systems, 17(1): 1-31. March 1992.

[7] Jim Gray. The Benchmark Handbook. Morgan Kaufmann, San Mateo, CA, 1993

[8] Naphtali Rishe. Database Design: the semantic modeling approach. McGraw-Hill, 1992, 528 pp.

[9] W.B. Rubenstein, M.S. Kubicar, R.G.G Cattell. Benchmarking Simple Database Operations. Proceedings of the ACM SIGMOD International Conference on Management of Data, San Francisco, CA, 387-394. May 1987.

[10] Naphtali Rishe, Wei Sun, David Barton, Yi Deng, Cyril Orji, Michael Alexopoulos, Leonardo Loureiro, Carlos Ordonez, Mario Sanchez, Artyom Shaposhnikov. Florida International University High Performance Database Research Center. SIGMOD Record, 24 (1995), 3, pp. 71-76.

[11] Kim Shanley. History and Overview of the TPC. Transaction Processing Performance Council. http://tpc.org/articles/tpc.overview.history.1.html, February 1998.

[12] Michael Stonebraker. Object-Relational Database Systems: The Next Wave. Morgan Kaufmann, 1996.

[13] Alexander Vaschillo, A semantic paradigm for intelligent data access. Ph.D. Dissertation, Florida International University, 2000.

[14] D. DeWitt, P. Futtersack, D. Maier and F. Velez. A study of three alternative workstation-server architectures for object oriented database systems. Proceedings of the Sixteenth Very Large Data Bases Conference, Brisbane, Australia, 1990. pp. 107-121.

Acknowledgements

We wish to thank Victor Sivov for his work in running and tuning The Semantic Benchmark for relational databases and for his invaluable comments on the benchmark.