Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

367

W. Litwin H.-J. Schek (Eds.)

INRIA

Foundations of Data Organization and Algorithms

3rd International Conference, FODO 1989
Paris, France, June 1989
Proceedings

Springer-Verlag

This series reports new developments in computer science research and teaching – quickly, informally and at a high level. The type of material considered for publication includes preliminary drafts of original papers and monographs, technical reports of high quality and broad interest, advanced level lectures, reports of meetings, provided they are of exceptional interest and focused on a single topic. The timeliness of a manuscript is more important than its form which may be unfinished or tentative. If possible, a subject index should be included. Publication of Lecture Notes is intended as a service to the international computer science community, in that a commercial publisher, Springer-Verlag, can offer a wide distribution of documents which would otherwise have a restricted readership. Once published and copyrighted, they can be documented in the scientific literature.

Manuscripts

Manuscripts should be no less than 100 and preferably no more than 500 pages in length.
They are reproduced by a photographic process and therefore must be typed with extreme care. Symbols not on the typewriter should be inserted by hand in indelible black ink. Corrections to the typescript should be made by pasting in the new text or painting out errors with white correction fluid. Authors receive 75 free copies and are free to use the material in other publications. The typescript is reduced slightly in size during reproduction; best results will not be obtained unless the text on any one page is kept within the overall limit of 18 x 26.5 cm (7 x 10½ inches). On request, the publisher will supply special paper with the typing area outlined.

Manuscripts should be sent to Prof. G. Goos, GMD Forschungsstelle an der Universität Karlsruhe, Haid- und Neu-Str. 7, 7500 Karlsruhe 1, Germany, Prof. J. Hartmanis, Cornell University, Dept. of Computer Science, Ithaca, NY/USA 14853, or directly to Springer-Verlag Heidelberg.

Springer-Verlag, Heidelberger Platz 3, D-1000 Berlin 33
Springer-Verlag, Tiergartenstraße 17, D-6900 Heidelberg 1
Springer-Verlag, 175 Fifth Avenue, New York, NY 10010/USA
Springer-Verlag, 37-3, Hongo 3-chome, Bunkyo-ku, Tokyo 113, Japan

ISBN 3-540-51295-0
ISBN 0-387-51295-0

Efficient Organization of Semantic Databases

Naphtali Rishe

School of Computer Science
Florida International University —
The State University of Florida at Miami
University Park, Miami, FL 33199

This paper argues that semantic data models can have potentially more efficient implementations than the conventional data models. As a step towards realization of this potential, the paper proposes an efficient storage structure for semantic databases.

1. Introduction

Since [Abrial-74], many semantic data models have been studied in the Computer Science literature. Although somewhat different in their terminology and their selection of tools used to describe the semantics of the real world, they have several common principles:

- The entities of the real world are represented in the database in a manner transparent to the user. (Unlike that, in the relational model the entities are represented by the values of keys of some tables; in the network model the entities are represented by record occurrences.) Hereinafter, the user-transparent representations of real-world entities are referred to as "abstract objects". The "concrete objects", or "printable values", are numbers, character strings, *etc*. The concrete objects have conventional representations on paper and in the computer.
- The entities are classified into types, or categories, which need not be disjoint. Meta-

This research has been supported in part by a grant from Florida High Technology and Industry Council

relations of inclusion are defined between the categories.

- Logically-explicit relationships are specified among abstract objects (*e.g.*, "person p1 is the mother of person p2") and between abstract objects and concrete objects (*e.g.*, "person p1 has first name 'Jack'"). There are no direct relationships among the concrete objects. In most semantic models, only binary relations are allowed, since higher order relations do not add any power of semantic expressiveness ([Bracchi-76], [Rishe-87-RM], [Rishe-88-DDF]), but do decrease the flexibility of the database and representability of partially-unknown information, and add complexity and potential for logical redundancy ([Rishe-88-DDF]).

The advantages of the semantic models versus the Relational and older models with respect to database design, database maintenance, data integrity, conciseness of languages, and ease of DML programming are known [Rishe-88-DDF]. This paper advocates the potential of semantic models to have efficient implementation.

Until now, several semantic data models have been implemented as interfaces to database managements systems in other data models, e.g., the relational or the network model [Tsur/Zaniolo-84]. (However, there are less typical, direct implementations, *e.g.* [Lien-81], [Chan-82], [Benneworth-81].) The efficiency of an interface implementation is limited to that of the conventional DBMS, and is normally much worse due to the interface overhead. The direct implementations are also commonly believed to have to be less time-efficient than the conventional systems, as a trade-off for the extra services that the semantic databases should provide. However, this author contends that the semantic models have potential for much more efficient implementation than the conventional data models. This is due to two reasons:

- All the physical aspects of representation of information by data are user-transparent in the semantic models. This creates greater potential for optimization: more things may be changed for efficiency considerations, without affecting the user programs. The Relational Model has more data independence than the older models. For example, the order of rows in the tables (relations) is transparent to the user. The semantic models have even more user-transparency. For example, the representation of real-world entities by printable values is transparent to the user. One may recall that not long ago the Relational Model was criticized as less efficient than the Network and Hierarchical models. However, it is clear now that optimizing relational database systems have potential of much higher efficiency than the network and hierarchical systems due to the data independence of the relational model.

- In the semantic models, the system knows more about the meaning of the user's data and about the meaningful connections between such data. This knowledge can be utilized to organize the data so that meaningful operations can be performed faster at the expense of less meaningful or meaningless operations.

In this paper, I use the Semantic Binary Model (SBM) [Rishe-88-DDF], a descendant of the model proposed in [Abrial-74]. This model does not have as rich an arsenal of tools for semantic description as can be found in some other semantic models, e.g. the IFO model [Abiteboul/Hull-84], SDM [Hammer/McLeod-81] (implementation [Jagannathan-88]), the Functional Model [Shipman-81] (implementation [Chan-82]), SEMBASE [King-84], NIAM ([Nijssen-81], [Nijssen/VanBekkum-82], [Leung/Nijssen-87]), GEM [Tsur/Zaniolo-84], TAXIS [Nixon-87], or the semi-semantic Entity-Relationship Model [Chen-76]. Nevertheless, the SBM has a small set of sufficient simple tools by which all the semantic descriptors of the other models can be constructed. This makes SBM easier to use for the novice, easier to implement, and usable for delineation of the common properties of the semantic models. The results of this paper are practically independent of the choice of a particular semantic model, and therefore they apply to almost all other semantic models.

The semantic binary model represents the information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects. The graphical database schema and the integrity constraints determine what sets of facts are meaningful, i.e. can comprise an instantaneous database (the database as may be seen at some instance of time.)

Formal semantics of the semantic binary model is defined in [Rishe-87-DS] using the methodology proposed in [Rishe-86-DN]. The syntax and informal semantics of the model and its languages (data definition languages, 4-th generation data manipulation languages, non-procedural languages for queries, updates, specification of constraints, userviews, *etc.*) are given in [Rishe-88-DDF]. A non-procedural semantic database language of maximal theoretically-possible expressive power is given in [Rishe-86-PS]. (In this language, one can specify every computable query, transaction, constraint, *etc.*)

The following section proposes an efficient storage structure for the Semantic Binary Model.

2. STORAGE STRUCTURE

2.1. Abstracted level

Every abstract object in the database is represented by a unique integer identifier. The categories and relations of the schema are also treated as abstract objects and hence have unique identifiers associated with them. Information in the database can then be represented using two kinds of facts, denoted xC and xRy, where x is the identifier associated with an abstract object, C and R are the identifiers associated with a category or a relation respectively, and y is either an identifier corresponding to an abstract object or a concrete object (a number or a text string). "xC" indicates that the object x belongs to the category C. "xRy" indicates that the object x is associated with the object y by the relation R. Logically, the instantaneous database is a set of such facts.

2.2. Goals

2.2.1. Efficiency of retrieval requests

At the intermediate level of processing queries and program retrieval requests, the queries are decomposed into *atomic retrieval operations* of the types listed below. The primary goal of the physical file structure is to allow a very efficient performance for each of the atomic requests. Namely, *each atomic retrieval request normally requires only 1 disk access*, provided the output information is small enough to fit into one block. When the output is large, the number of blocks retrieved is close to the minimal number of blocks needed to store the output information.

1. **aC** — Verify the fact aC. (For a given abstract object a and category C, verify whether the object a is in the category C.)
2. **aRy** — Verify the fact aRy.
3. **a?** — For a given abstract object a, find all the categories to which a belongs.
4. **?C** — For a given category, find its objects.
5. **aR?** — For a given abstract object a and relation R, retrieve all y such that aRy. (The objects y may be abstract or concrete.)

6. **?Ra** For a given abstract object a and relation R, retrieve all abstract objects x such that xRa.

7. **a?+a??+??a** Retrieve all the immediate information about an abstract object. (I.e., for a given abstract object a, retrieve all of its direct and inverse relationships, that is, the relations R and objects y such that aRy or yRa; and the categories to which a belongs.)

   (Although this request can be decomposed into a series of requests of the previous types, we wish to be able to treat it separately in order to ensure that the whole request normally be performed in a single disk access. This will also allow a single-access performance of requests which require several, but not all, of the facts about an object, e.g. a query to find the first name, the last name, and the age of a given person.)

8. **?Rv** For a given relation (attribute) R and a given concrete object (value) v, find all abstract objects x such that xRv.

9. **?R[v1,v2]** For a given relation (attribute) R and a given range of concrete objects $[v_1, v_2]$, find all objects x and v such that xRv and $v_1 \le v \le v_2$. (The comparison "≤" is appropriate to the type of v.)

2.2.2. Efficiency of update transactions

Efficient performance of update transactions is required, although more than one disk access per transaction is allowed.

A transaction is a set of interrelated update requests to be performed as one unit. Transactions are generated by programs and by interactive users. A transaction can be generated by a program fragment containing numerous update commands, interleaved with other computations. However, until the last command within a transaction is completed, the updates are not physically performed, but rather accumulated by the DBMS. Upon completion of the transaction the DBMS checks its integrity and then physically performs the update. The partial effects of the transaction may be inconsistent. Every program and user sees the database in a consistent state: until the transaction is committed, its effects are invisible.

A completed transaction is composed of a set of facts to be deleted from the database, a set of facts to be inserted into the database, and additional information needed to verify that there is no interference between transactions of concurrent programs. If the verification produces a positive result, then the new instantaneous database is: ((the-old-instantaneous-database) − (the-set-of-facts-to-be-deleted)) ∪ (the-set-of-facts-to-be-inserted).

2.3. Solution: a file structure achieving the goals

The following file structure supports the above requirements. The entire database is stored in a single file. This file contains all the facts of the database (xC and xRy) and also additional information described below and called inverted facts. The file is maintained as a B-tree. The variation of the B-tree used here allows both sequential access according to the lexicographic order of the items comprising the facts and the inverted facts, as well as random access by arbitrary prefixes of such facts and inverted facts.

The facts which are close to each other in the lexicographic order reside close in the file. (Notice, that although technically the B-tree-key is the entire fact, it is of varying length and on the average is only several bytes long, which is the average size of the encoded fact xRy. The total size of the data stored in the index-level blocks of the B-tree is less than 1% of the size of the database: *e.g.* each 10,000-byte data block may be represented in the index level by its first fact --5 bytes — and block address — 3 bytes — which would amount to 0.08% of the data block. Thus, all the index blocks will fit even into relatively small main memory.)

The file contains the original facts and additionally the following "inverted facts":

1. In addition to xC, we store its inverse $\bar{C}x$. ($\bar{C}$ is the system-chosen identifier to represent the inverse information about the category C. For example, it can be defined as $\bar{C} = 0\text{–}C$.) (If a category C_1 is a subcategory of category C_2, an object a belongs to C_1 and, thus, also to C_2, then we chose to store both inverted facts $\bar{C}_1 a$ and $\bar{C}_2 a$. When the user requests the deletion of the fact aC_2, it triggers automatic deletion of the facts aC_1, $\bar{C}_1 a$, and $\bar{C}_2 a$ in order to guarantee consistency.)

2. In addition to xRv, where v is a concrete object (a number, a string, or a value of another type), we store $\bar{R}vx$. Thus, the range query "?R[v1,v2]" is satisfied by all and only the inverted facts which are positioned in the file between $\bar{R}v_1$ and $\bar{R}v_2\underline{\text{HighSuffix}}$. (HighSuffix is a suffix which is lexicographically greater than any other possible suffix.) Thus, the result will most probably appear in one physical block, if it can fit into one block.

3. In addition to xRy, where both x and y are abstract objects, we store $y\bar{R}x$. Thus, for any abstract object x, all its relationships xRy, xRv, zRx, and xC can be found in one place in the file: the regular and inverted facts which begin with the prefix x. (The infixes are: categories for xC, relations for xRy and xRv, and inverse relations $x\bar{R}z$ from which we find z such that zRx.)

Notice that facts xRa and xRv (x and a are abstract objects, v is a value) are inverted dissimilarly. This is because we have different types of atomic retrieval requests concerning abstract and concrete objects:

- There are range queries with concrete objects, e.g. "Find all persons salaried between $40,000 and $50,000". In such queries we know the identifier of the relation and partial information about the value. Therefore we need to use the inverted facts with $\bar{R}$ as the prefix. There are no range queries with abstract objects.
- On the other hand, we have multiple-fact retrievals about an abstract object, e.g. "Find all the immediate information about a given person p" (while such a request about a concrete object would be meaningless: "Find all the information about the number 5" makes no sense, as oposed to a meaningful query "Find information about item(s) whose price is $5".) Here we know the object, but do not know the identifiers of the inverted relations. We need to cluster together all the inverted relations of one object. Therefore, the inverted relation should appear in the infix.

The sorted file is maintained in a structure similar to a B-tree. The "records" of the B-tree are the regular and inverted facts. The records are of varying length. The B-tree-keys of the "records" are normally the entire B-tree-records, *i.e.* facts, regular and inverted. (An exception from this is when the record happens to be very long. The only potentially long records represent facts xRv where v is a very long character string. We employ a special handling algorithm for very long character strings.) Access to this B-tree does not require knowledge of the entire key: any prefix will do. All the index blocks of the B-tree can normally be held in cache.

At the most physical level, the data in the facts is compressed to minimal space. Also, since many consecutive facts share a prefix (e.g. an abstract object identifier) the prefix need not be repeated for each fact. In this way the facts are compressed further. The duplication in the number of facts due to the inverses is 100%, since there is only one inverse per each original fact (with a rare exception of the storage of redundant inverses of supercategories as described in (1)). The B-tree causes additional 30% overhead. (This overhead is because in a B-tree the data blocks are only 75% full on the average, though

this can be improved by periodical reorganization. The overhead for the index blocks of the B-tree is no more than 1-2% since they contain only one short fact per every data block.) The total space used for the database is therefore only about 160% more than the amount of information in the database, i.e. the space minimally required to store the database in the most compressed form with no regard to the efficiency of data retrieval or update. Thus, the data structure described herein is more efficient in space and time than the conventional approach with separate secondary index files for numerous fields.

No separate index files are needed in the file structure proposed in this paper. The duplication of data (*i.e.* inverted relations) together with the primary sparse index which is a part of the B-tree effectively eliminate the need for secondary (dense) indices. Furthermore, it eliminates the horrendous I/O operations caused by sequentially retrieving along a secondary index, since the sequence of information represented by our primary sparse index is also stored in consecutive physical locations.

2.4. Secondary aspects of the file structure

Efficiency (time and storage) and flexibility of the file structure are enhanced by compression and encoding of the facts and the items therein, including: abstract objects; character strings of unlimited length; numbers with no global minimum, maximum or precision; identifiers for categories, relations, and their inverses.

A special technique [Rishe-88-CM] is employed to encode the numbers in the facts of the file so that:

1. Bitwise lexicographic comparison of the encodings coincides with the meaningful comparison of numbers. Thus, if n_1 is encoded by a byte string $b_1^1 b_2^1 b_3^1$ and n_2 is encoded by a byte string $b_1^1 b_2^2 b_3^2 b_4^2$, where $b_2^1 > b_2^2$, then $n_1 > n_2$. The standard representations of numbers do not allow bitwise comparison. (Consider, for example, the representation of floating point numbers by mantissa and exponent.)
2. There is no limit on arbitrarily large, arbitrarily small, or arbitrarily precise numbers. We wish to be able to compare and store in a uniform format integers, real numbers, numbers with very many significant digits, and numbers with just a few significant digits. We do not wish to set a limit on the range of the data at the time of the design or creation of the file. For example, the number π truncated after first 1000 digits is a very precise number (1000 significant digits). The number $10^{10^{10}}$ is large (10 billion decimal digits, but only one significant digit). We use the same format convention to represent both numbers. (The length of the representation is approximately

proportional to the number of significant digits: the representation of the second number is only a few bytes long, the representation of the first number is 425 bytes long.)

3. Every number bears its own precision, *i.e.* the precision is not uniform throughout the database. (This allows to treat integers, reals, values of different attributes with different precisions, in a uniform way in one file in the database.)
4. The encodings are of varying length and are about maximally space efficient with respect to their informational content. For example, the numbers 3,000,000 (with precision 500,000), integer 5 (precision 0.5), 0.000,000,000,000,000,7 (with precision 0.000,000,000,000,000,05) should require only a few bits each, while the number 12345678.90 should require many more bits. The number of bits in a number's representation is approximately equal to the amount of information in that number.
5. No additional byte(s) are required to store the length of the encoded representation or to delimit its end: the representation should contain enough information within itself so that the decoder would know where the representation of one number ends and of the next begins (within the same record in the file.) The absence of delimiters gives an additional saving in space, and also facilitates handling of records.
6. The representation of numbers is one-to-one. For example, there may not be several representation for 0, like 0.00, -0.0, 0E23, 0E0.
7. The encoding and decoding should be relatively efficient (linear in the length of the data string), but they need not be as efficient as comparisons. The database system can handle encoded numbers in all the internal operations, and translate them only on input/output from/to the external user.

3. Comparison to performance of implementations of the Relational Model

The system proposed herein is not less efficient, and normally more efficient, in both time and storage space than the relational model's implementations with multiple dense indices.

Let us consider a simple relational database composed of one relation T with attributes $A_1, A_2, \ldots, A_n$. Let us assume that for each j there are queries of the type

$$\text{get } A_i \text{ where } A_j=c \qquad \text{(Q1)}$$

and that each of those queries is required to be performed in a reasonable time.

The Relational model is technically a subset of the Semantic Binary Model. Specifically, the above relational schema is viewed in the Semantic Binary Model as a category T and relations A_i between the objects of T and values.

To assure reasonable time performance in the Relational model for each of the above queries, we need a dense index on each of the attributes A_i. There are n index files (or n indices combined in one file in some implementations.) The total size of the indices thus exceeds the size of the table T itself. Therefore the space overhead in the Relational model is greater than 100% and, thus, is greater than the space overhead in the proposed semantic implementation. Also, in the semantic implementation there is only one physical file, while there are many physical files in the relational implementations (and in some implementations there are as many files as $number\_of\_tables \times (1+number\_of\_attributes\_per\_table)$. The management of multiple files is not only a hassle but also contributes to additional space overhead due to allocation of growth areas for each file.

With respect to the time required to solve the simple queries of type Q1, it is the same in the best relational implementations and in the proposed semantic implementation. Namely, the time is

$$(1+number\_of\_values\_in\_the\_output) \times time\_to\_retrieve\_one\_block$$

(In the relational implementation, there will be one visit to the dense index on A_j, and for every $A_j=c$ found there, there will be one random access to the main table. In the semantic implementation, first the sub-query $?A_jc$ will be solved, and then for every match x found the sub-query $xA_i?$ will be evaluated.)

If in Q1 we desired to print many attributes A_i instead of just one, the same time results would be obtained in both implementations. Notice that in the semantic implementation proposed herein all the immediate information of an object, including all its attributes, is clustered together.

Now let us consider updates. Insertion of a row into the relational table takes replacement of one block in the main table and n blocks in the dense indices. In the semantic implementation there is insertion of the primary facts about the new object ob $obA_1c_1, \ldots, obA_nc_n$ (all the primary facts will appear in contiguous storage in one block) and n inverse facts in possibly different n blocks. Thus, here, as well as in the other types of simple updates, the performance of the semantic implementation is not worse than tha

of the relational implementations supporting efficiency of queries.

The advantages in the semantic implementation's performance become even more significant for more complex queries and updates. Though the detailed analysis of these is beyond the space-limit of this paper, I would like to mention that, for example, queries requiring natural join in the relational implementations would be more efficient in the semantic implementation because there are direct explicit relationships between the categories instead of relationships represented implicitly by foreign key in the Relational Model. The gap in performance between the faster semantic implementation and the relational implementations increases even more when the relational keys are composed of more than one attribute and when the relationships between the tables are many-to-many, which requires an extra table to represent the many-to-many relationship in the relational implementations. The gap increases with the number of joins in the query. In general, the more complex the query is the greater is the advantage in the efficiency of the proposed semantic implementation versus the relational implementations.

Of course, there are also major efficiency advantages in the semantic implementation in support of semantic complexities of the real world, which are very awkwardly and inefficiently implemented in the relational implementations. These complexities include intersecting categories, sub-categories, categories with no keys, varying-length attributes, missing ("null") values, multiple values, etc.

4. CONCLUSION

We have implemented this data structure in a prototype DBMS at the University of California, Santa Barbara ([Vijaykumar-87], [Jain-87]). Our implementation allows single-processor multi-user parallel access to the database. Optimistic concurrence control is used.

Although the best results are obtained from our DBMS for the Semantic Binary Model, it can also be used efficiently with all other major semantic and conventional database models. This is due to the fact that the Relational, Network, and Hierarchical data models are technically subsets of the Semantic Binary Model (as shown in [Rishe-88-DDF]).

Currently, at Florida International University, we are working on a project, financed by the state government, to extend our semantic DBMS implementation into a massively-parallel very-high-throughput database machine [Rishe-88-AMPDM], to be

composed of many (thousand[s]) processors, each equipped with a permanent storage device and a large cache memory. Our analysis has shown that the proposed file structure greatly increases the parallelism in the operations of the DBMS, which can be utilized by large-scale parallel machines.

Acknowledgment

The author gratefully acknowledges the advice of Narayanan Vijaykumar, Li Qiang, Nagarajan Prabhakaran, Doron Tal, and David Barton.

References

[Abiteboul/Hull-84] S. Abiteboul and R. Hull. "IFO: A Formal Semantic Database Model", Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1984.

[Abrial-74] J.R. Abrial, "Data Semantics", in J.W. Klimbie and K.L. Koffeman (eds.), *Data Base Management*, North Holland, 1974.

[Benneworth-81] R.L. Benneworth, C.D. Bishop, C.J.M. Turnbull, W.D. Holman, F.M. Monette. "The Implementation of GERM, an Entity-relationship Data Base Management System". Proceedings of the Seventh International Conference on Very Large Data Bases. (Eds. C. Zaniolo & C. Delobel.) IEEE Computer Society Press, 1981. (pp 465-477)

[Bracchi-76] Bracchi,G., Paolini, P., Pelagatti, G. "Binary Logical Associations in Data Modelings". In G.M. Nijssen (ed.), *Modeling in Data Base Management Systems*. IFIP Working Conference on Modeling in DBMS's, 1976.

[Chan-82] Chan,A., Danberg,Sy, Fox,S., Lin,W-T.K., Nori,A., and Ries,D.R. "Storage and Access Structures to Support a Semantic Data Model" Proceedings of the Eighth International Conference on Very Large Data Bases. IEEE Computer Society Press, 1982.

[Chen-76] P. Chen. "The Entity-relationship Model: Toward a unified view of data." *ACM Trans. Databas Syst. 1*, 1, 9-36.

[Hammer/McLeod-81] M. Hammer and D. McLeod. "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems*, Vol. 6, No. 3, pp. 351-386, 1981.

[Jagannathan-88] D. Jagannathan, R.L. Guck, B.L. Fritchman, J.P. Thompson, D.M. Tolbert. "SIM: A Database System Based on Semantic Model." *Proceedings of SIGMOD International Conference on Management of Data.* Chicago, June 1-3, 1988. ACM-Press, 1988.

[Jain-87] A. Jain. Design of a Binary Model Based DBMS and Conversion of Binary Model Based Schema to an Equivalent Schema in Other Major Database Models. M.S. Thesis, University of California, Santa Barbara, 1987.

[King-84] R.King. "SEMBASE: A Semantic DBMS." Proceedings of the First Workshop on Expert Database Systems. Univ. of South Carolina, 1984. (pp. 151-171)

[Leung/Nijssen-87] C.M.R. Leung and G.M. Nijssen. From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema, Aust. Comput. J., Vol. 19, No. 2.

[Lien-81] Y.E. Lien, J.E. Shopiro, S. Tsur "DSIS — A Database System with Interrelational Semantics". Proceedings of the Seventh International Conference on Very Large Data Bases. (Eds. C. Zaniolo & C. Delobel.) IEEE Computer Society Press, 1981. (pp 465-477)

[Nijssen-81] G.M. Nijssen "An architecture for knowledge base systems", Proc. SPOT-2 conf., Stockholm, 1981.

[Nijssen/VanBekkum-82] G.M.A. Nijssen and J. Van Bekkum. "NIAM - An Information Analysis Method", in Information Systems Design Methodologies: A Comparative Review, T.W. Olle, et al. (eds.), IFIP 1982, North-Holland.

[Nixon-87] B. Nixon, L. Chung, I. Lauzen, A. Borgida, and M. Stanley. Implementation of a compiler for a semantic data model: Experience with Taxis." In *Proceedings of ACM SIGMOD Conf.* (San Francisco), ACM, 1987.

[Rishe-86-DN] N. Rishe. "On Denotational Semantics of Data Bases." *Mathematical Foundations of Programming Semantics.* Proceedings of the International Conference on Mathematical Foundations of Programming Semantics, April 1985, Manhattan, Kansas (ed. A. Melton), Lecture Notes in Computer Science, vol. 239. Springer-Verlag, 1986. (pp 249-274.)

[Rishe-86-PS] N. Rishe. "Postconditional Semantics of Data Base Queries." *Lecture Notes in Computer Science, vol. 239 (Mathematical Foundations of Programming Semantics*, ed. A. Melton), pp 275-295. Springer-Verlag, 1986.

[Rishe-87-DS] N. Rishe, *Database Semantics.* Technical report TRCS87-002, Computer Science Department, University of California, Santa Barbara, 1987.

[Rishe-87-RM] N. Rishe. "On Representation of Medical Knowledge by a Binary Data Model." *Journal of Mathematical and Computer Modelling*, vol. 8, 1987. (pp. 623-626)

[Rishe-88-AMPDM] N. Rishe, D. Tal, and Q. Li. "Architecture for a Massively Parallel Database Machine" *Microprocessing and Microprogramming.* The Euromicro Journal. 1988, in press.

[Rishe-88-CM] N. Rishe. "A Compact Monotonic Universal Encoding of Numbers" Proceedings of the 26th Annual Conference of Southeast Region of the Association for Computing Machinery, 1988.

[Rishe-88-DDF] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology.* Prentice-Hall, Englewood Cliffs, NJ, 1988. 436 pages. ISBN 0-13-196791-6.

[Shipman-81] D.W. Shipman. "The Functional Data Model and the Data Language DAPLEX", *ACM Transactions on Database Systems*, v. 6, no. 1, 140-173, 1981.

[Tsur/Zaniolo-84] S. Tsur, C. Zaniolo. "An implementation of GEM — supporting a semantic data model on a relational backend." In *Proc. ACM SIGMOD Intl. Conf. on Management of Data, May 1984.*

[Vijaykumar-87] N. Vijaykumar. Toward the Implementation of a DBMS based on the Semantic Binary Model. M.S. Thesis, University of California, Santa Barbara, 1987.