89-SD

SIXTH SYMPOSIUM ON MICROCOMPUTER AND MICROPROCESSOR APPLICATIONS

FROM
17th OCTOBER TO 19th OCTOBER

1989

BUDAPEST—HUNGARY

Vol. I.

Semantic database management: from microcomputers to massively parallel database machines

Naphtali Rishe

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

This paper describes the use and implementation of the semantic database models, and of the Semantic Binary Model in particular, on sequential and parallel computer architectures. The paper discusses logical aspects of semantic models, compares features of semantics models to older database models, particularly the Relational Model, and describes an efficient storage structure for the semantic model. The paper concludes with a discussion of the massively-parallel semantic database machine LSDM.

1. Introduction

A database model is a convention and method of organization and usage of databases. The predominant database models of the seventies were the Hierarchical and Network models. The predominant model of the late eighties is the Relational Model. It is believed by most experts in the field that the models used in the nineties will be models of the next generation, namely *semantic* database models.

The primary purpose of the semantic models is to provide a simple, natural, computer-independent, flexible, and non-redundant specification of information and of the logical structure of the user's enterprise. The meaning of the user's information is analyzed and stored in the computer. The computer is made to comprehend and treat the information in the way similar to that of the human user. The human-computer gap is narrowed; the interaction is made easier; harder and more complex tasks can be specified by the human user and performed by the computer.

Semantic models offer major advantages versus the Relational and older models with respect to database design, database maintenance, conciseness of languages, reliability and consistency of data, ease of application programming, and friendliness of the end-user services. In addition to the above advantages, it has been recently proved in [Rishe-89-EO] that the semantic models have potential for much more efficient (faster response) implementation than the conventional data models.

A representative of the semantic models is the Semantic Binary Model (SBM) ([Rishe-87-RM], [Rishe-88-DDF], [Rishe-89-DDS]). Its advantage over the other proposed semantic models is that SBM has a small set of sufficient simple tools by which all the semantic aspects of any application can be described. This makes SBM easier to use for the novice, *i.e.* friendlier for the user, and also easier to implement.

2. The Semantic Binary Model

The semantic models provide a simple, natural, data-independent, flexible, and logically non-redundant specification of information and its semantic aspects. Many semantic models have been surveyed in [Hull/King-87] and [Peckham/Maryanski-88]. Although somewhat differing in their terminology and their selection of tools used to describe the semantics of the real world, the semantic models have several common principles:

- The entities of the real world are represented in the database in a manner transparent to the user. 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 computer representations.
- The entities are classified into types, or categories, which need not be disjoint. Meta-relations of inclusion are defined between the categories, implying inheritance of properties.
- Logically-explicit relationships are specified among abstract objects (*e.g.*, "person p_1 is the mother of person p2") and between abstract and concrete objects (*e.g.*, "person p_1 has first name 'Jack'"). There are no direct relationships among the concrete objects. In most semantic models, only binary relations are allowed, since they provide full power of semantic expressiveness. Higher order relations do not add any power of semantic expressiveness (in [Bracchi *et al*-76], [Rishe-87-RM], and [Rishe-88-DDF], the advantages of the binary relations versus high-order relationships were shown.)

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 have been shown, *e.g.*, in [Rishe-88-DDF].

Typically, semantic data models are implemented as interfaces to database management systems in other data models, e.g., the relational or the network model. (Although, there are less typical, direct implementations, *e.g.* [Lien *et al*-81], [Chan *et al*-82], [Benneworth *et al*-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. However, the semantic models have potential for much more efficient implementation than the conventional data models. This is due to two facts:

- All the physical aspects of representation of information 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 yet more user-transparency. For example, the representation of real-world entities by printable values is transparent to the user.
- 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.

The Semantic Binary Model (SBM) [Rishe-88-DDF] is 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 [UNISYS-87]), the Functional Model [Shipman-81] (implementation [Chan *et al*-82]), SEMBASE [King-84], NIAM ([Nijssen-81], [Nijssen/VanBekkum-82], [Leung/Nijssen-87]). 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 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.)

Example 2-1.

Consider a database of which the following is a sub-schema:

- Category *COMPANY*
- Category *PRODUCT*
- Relation *company-name* from *COMPANY* to the category of values *String* (*1:1*)
- Relation *description* from *PRODUCT* to the category of values *String* (*1:1*)
- Relation manufactures from *COMPANY* to *PRODUCT* (*m:m*)

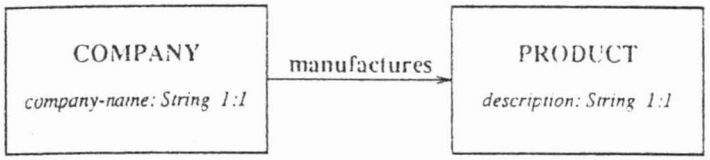

Figure 2-1. A sub-schema of a database

The following set of facts can be a part of a logical instantaneous database:

1. object1 **COMPANY**
2. object1 ***COMPANY-NAME*** 'IBM'
3. object1 ***MANUFACTURED*** object2
4. object1 ***MANUFACTURED*** object3
5. object2 ***PRODUCT***
6. object2 ***DESCRIPTION*** 'IBM/SYSTEM-2'
7. object3 ***PRODUCT***
8. object3 ***DESCRIPTION*** 'MONOCHROMATIC- MONITOR'

Example 2-2.

As a larger example, the schema of Figure 2-2 describes some activities of a Dining Club, including the following information:

PATRON — a member of the club. For each patron it is known what are his two most favorite tables in the club. Each table has one or more names, but no two tables have the same name.

PERSON — any person who is a patron or waiter or both.

CHARGE ACCOUNT — account to which meals can be charged. There is no permanent relationship between an account and a patron, since there may be party's accounts, complimentary promotional accounts, patrons' consortium accounts, employers' accounts, etc. Ad hoc relationships between patrons and accounts are established when the patron's meal is charged to the account. The club does not accept cash for meals. The charge for every meal is always $10, therefore the money amounts do not appear in the database explicitly.

PARTY — an ad hoc group of patrons eating on a particular date during the same shift, served by the same waiter, and charging to the same account.

The following are some of the concepts comprising the schema:

- Category *TABLE*
- Category *PERSON*
- Subcategory *PATRON* of the category *PERSON*
- Subcategory *WAITER* of the category *PERSON*
- Category *SHIFT*
- Category *ACCOUNT*
- Category *PARTY*
- Category *MEAL*
- Relation **serves** from *WAITER* to *TABLE* (*m:m*)
- Relation *name* from *TABLE* to the category of values *String* (*1:m*) (A table may have several names, but every name is unique.)
- Relation *last-name* from *PERSON* to the category of values *String* (*m:1*)

The 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, 4-th generation data manipulation, 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 [Rishe-86-PS]. In this language, one can specify computable queries, transactions, constraints, etc.

3. Storage Structure

In this section we discuss an efficient storage structure for the Semantic Binary Model. This storage structure results in very efficient performance of simple queries.

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. The collection of facts forming the database is represented by a file structure which insures approximately 1 disk access to retrieve any of the

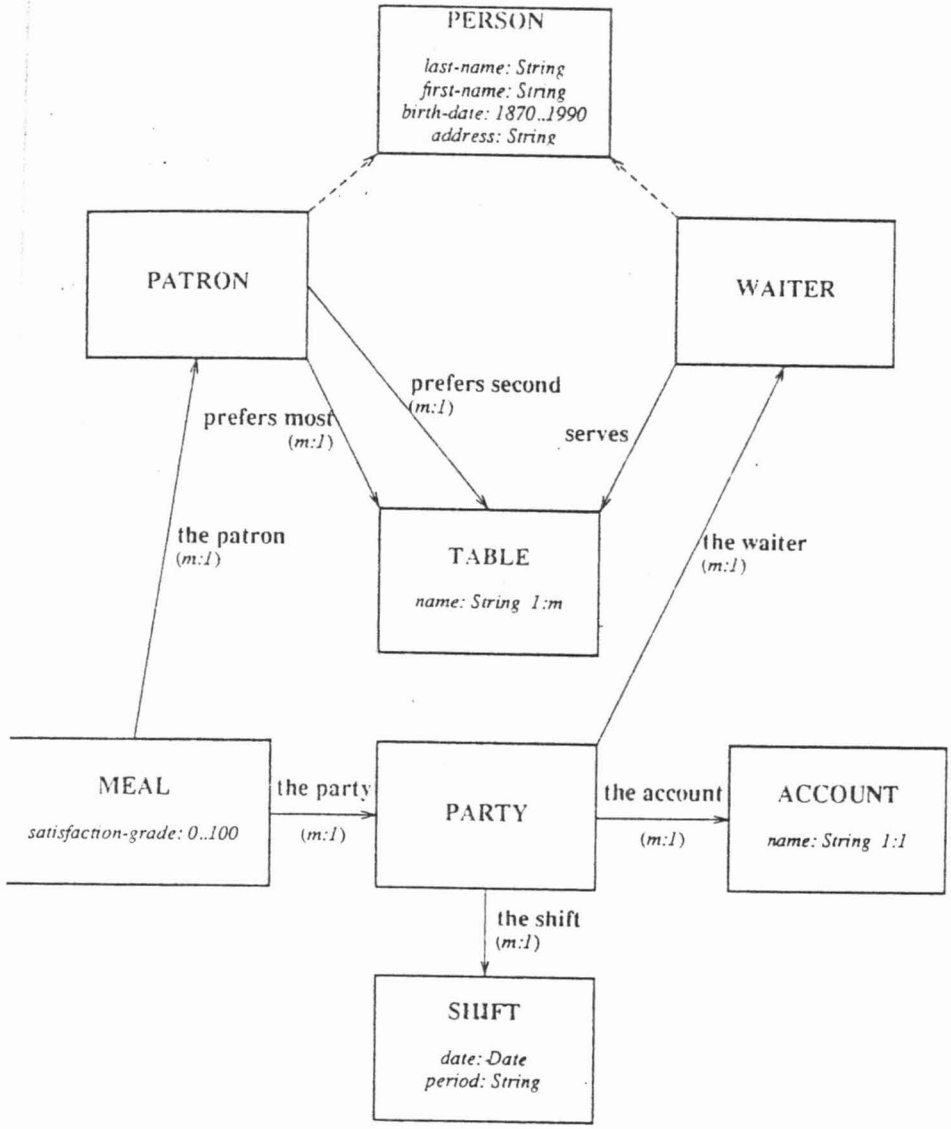

Figure 2-2. A binary schema for a dining-club application.

following:

1. For a given abstract object x, verify/find to what categories does the object belong.

2. For a given category, find its objects.
3. For a given abstract object x and relation R, retrieve all/certain y such that xRy.
4. For a given abstract object y and relation R, retrieve all/certain abstract objects x such that xRy.
5. For a given abstract object x, retrieve all of, or several of, its direct and/or inverse relationships, *i.e.* relations R and objects y such that xRy or yRx. (Note: despite the bulkiness of this operation, it can normally be performed in just one disk access.)
6. For a given relation (attribute) R and a given concrete object y, find all abstract objects such that xRy.
7. For a given relation (attribute) R and a given range of concrete objects $[y_1, y_2]$, find all objects x and y such that xRy and $y_1 \le y \le y_2$.

The above operations are called elementary queries, *i.e.* the simplest queries as far as the user is concerned. A file structure, which satisfies the above requirements, i.e. approximately one access to disk to perform any elementary query, is shown in [Rishe-89-EO]. Here we briefly describe it.

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 10,000B 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 $\overline{C}x$. ($\overline{C}$ is the system-chosen identifier to represent the inverse information about the category C. For example, it can be defined as $\overline{C} = 0\text{–}C$.) (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 $\overline{C_1}a$ and $\overline{C_2}a$. When the user requests the deletion of the fact aC_2, it triggers automatic deletion of the facts aC_1, $\overline{C_1}a$, and $\overline{C_2}a$.)
2. In addition to xRv, where v is a concrete object (a number, a string, or a value of another type), we store $\overline{R}vx$. 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 $\bar{R}v_{2*}$. 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.)

> *Example 3-1.*
>
> Consider the instantaneous database of Example 2-1. The additional inverted facts stored in the database are:
>
> 4. ***COMPANY***^{inv} object1
> 5. ***COMPANY-NAME***^{inv} 'IBM' object1
> 6. object2 ***MANUFACTURED***^{inv} object1
> 7. object3 ***MANUFACTURED***^{inv} object1
> 8. ***PRODUCT***^{inv} object2
> 9. ***DESCRIPTION***^{inv} 'IBM/SYSTEM-2' object2
> 10. ***PRODUCT***^{inv} object3
> 11. ***DESCRIPTION***^{inv} 'MONOCHROMATIC- MONITOR' object3

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 elementary queries 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, *e.g.*: "Find all the information about the number 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.

> *Example 3-2.*
>
> When the set of the original facts (Example 2.1) and the set of inverted facts (Example 2.2) are interleaved and lexicographically sorted, we obtain:

1. object1 COMPANY
2. object1 *COMPANY-NAME* 'IBM'
3. object1 *MANUFACTURED* object2
4. object1 *MANUFACTURED* object3
5. object2 *DESCRIPTION* 'IBM/SYSTEM-2'
6. object2 *PRODUCT*
7. object2 *MANUFACTURED*^{inv} object1
8. object3 *DESCRIPTION* 'MONOCHROMATIC- MONITOR'
9. object3 *PRODUCT*
10. object3 *MANUFACTURED*^{inv} object1
11. *COMPANY*^{inv} object1
12. *DESCRIPTION*^{inv} 'IBM/SYSTEM-2' object2
13. *DESCRIPTION*^{inv} 'MONOCHROMATIC- MONITOR' object3
14. *COMPANY-NAME*^{inv} 'IBM' object1
15. *PRODUCT*^{inv} object2
16. *PRODUCT*^{inv} object3

Example 3-3.

To answer the elementary query "Find all the information about object3 (including its direct and inverse relationships)", we find all the entries whose prefix is object3. These entries are clustered together in the sorted order.

To answer the elementary query "Find all objects manufactured by object1", we find all the facts whose prefix is object1_*MANUFACTURED*. ('_' denotes concatenation.) These entries are clustered together in the sorted order.

To answer the query to print the descriptions of the objects manufactured by the companies whose names are between 'IATA' and 'K-mart', we solve several elementary sub-queries, the first of which is:

1. Find the companies whose names are in the above range strings. (We search for inverted facts which are lexicographically between *COMPANY-NAME*^{inv}_'IATA' and *COMPANY-NAME*^{inv}_'K-mart'_HighSuffix.)

The sorted file is maintained in a B-tree structure. 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. Access to this B-tree does not require knowledge of the entire key: any prefix will do. The data level of the B-tree is stored on the disk, while the index levels of the B-tree, which are about 1% of the database [Rishe-89-EO], are assumed to be in cache. At the physical level, the data is compressed to minimal space.

The elementary queries are answered by physically adjacent facts. Using the above B-tree structure, we obtain normally just one disk-block access per elementary query (provided, the results fit in one block.)

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 super-categories as described in (1)). The B-tree causes additional 30% overhead. The total space used for the database is 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 [Rishe-89-EO]. 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. There are no separate index files in the file structure proposed in this paper.

The file will be partitioned in lexicographic order into many subfiles, each of which will reside on a separate disk and be controlled by a separate processor.

The above storage structure renders efficient implementation of semantic database on a wide range of single-processor computers, from microcomputers to mainframes. In the next section we will show how the above storage structure is mapped into parallel and massively architectures.

4. LSDM: A Semantic Database Machine

The LSDM [Rishe-et-al-89-AM] (Linear-throughput Semantic Database Machine) is a recent model of a multi-disk multi-processor database machine which offers massive parallelism and an optimized implementation for the semantic binary model. The architecture is designed to satisfy the following requirements:

- the computer performance should respond to the continuing increase in volume of queries and database size;
- it should be relatively easy to expand the system by adding additional processors and disks and the throughput is expected to increase linearly;
- it is necessary to have a degree of tolerance to faults in order to improve system reliability and safety;

dynamic load balancing is desired in order to share the work among the processors and to reduce idle time.

All the processors of LSDM are identical and may communicate via high speed communication channels that form together a hypercube-like network [Heller-85]. A schematic description of the architecture is shown in Figure 4.1. The figure depicts a small hypercube of 32 processors equipped with disks. The system can grow to thousands of processors. Further details on the LSDM architecture can be found in [Rishe *et al*-89-AM].

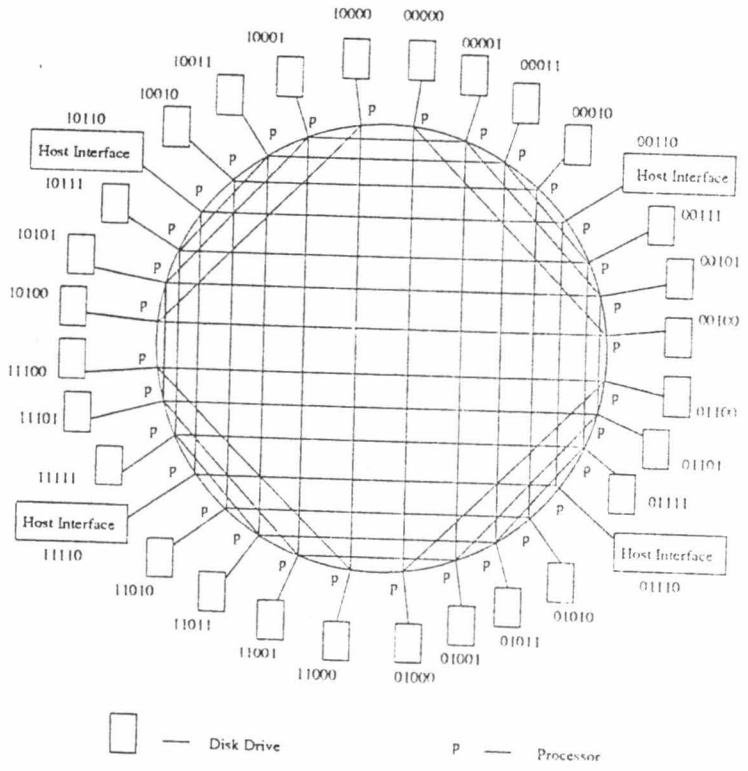

Figure 4-1. LSDM architecture

The database, including all the inverse facts, is represented by one logical file. This file is partitioned into many small fragments, each residing on a separate small disk. Each disk is associated with a fairly powerful processor. The number of disk-processor pairs is sufficient to accommodate the totality of the database. In order to minimize slow disk accesses, each disk-processor pair is associated with a large cache memory, allowing a semi-associative reference. Each processor can retrieve information from its local disk, perform the necessary

processing on the data and deliver the result to the user or to the other processors.

Every processor has a copy of the partitioning map of the entire database. Since the whole database is a lexicographically ordered file represented by a set of B-trees, the map only indicates for each node what is the lexicographically-maximal fact to be stored on its node. The map changes only when the database is re-partitioned. Special algorithms ensure that repartitioning is rare, inexpensive, and transparent to the system until all the shifting of data is complete [Rishe/Tal-89-PD].

Transactions and queries enter into the network through host interfaces. As illustrated in Figure 4.1, some of the processors serve as host processors. For every query and transaction the system designates a processor, called transaction/query manager or originator, which is charged with the distribution of the query/transaction and with producing a final result thereof. The stream of completed transactions proceeds in the LSDM through a set of filters: translation of user-views and subschemas, integrity verification, logging, data encoding, etc. Many of the filters can be implemented by separate processors. After that, the transaction is split into sub-transactions which will be performed on several disks by their processors. After performing the sub-transactions, the processors will respond to the transaction scheduler.

Queries come to the physical part of the DBMS in several streams, through filters. After initial translation and filtering, the queries go through multiplexer-integrator processors. The multiplexer-integrator splits a query into a set of parallel and/or sequential sub-queries to be performed by different processors. When the answers arrive, the results are integrated and sent to outer layers of the DBMS. While waiting for the answers, the multiplexer/integrator continues to work on other queries and send them to other disks.

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 *et al*-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 *et al*-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 *et al*-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.

[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.

[Heller-85] S. Heller. "Directed Cube Networks: A Practical Investigation", CSG Memo 253, M.I.T., July 1985.

[Hull/King-87] R. Hull and R. King. "Semantic Data Models." *ACM Computing Surveys. 20*, 3, 153-189.

[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 *et al*-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.

[Peckham/Maryanski-88] J. Peckham and F. Maryanski. "Semantic Database Modeling: Survey, applications, and research issues." *ACM Computing Surveys, 19*, 3, 201-260.

[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-DDF] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology.* Prentice-Hall, Englewood Cliffs, NJ, 1988. 456 pages. ISBN 0-13-196791-6.

[Rishe-89-DDS] N. Rishe. *Database Design: The Semantic Modeling Approach.* Prentice-Hall, Englewood Cliffs, NJ, accepted to appear in 1990, approx. 550 pages.

[Rishe-89-EO] N. Rishe. "Efficient Organization of Semantic Databases" Proceedings of the Third International Conference on Foundations of Data Organization, Paris, June 21-23, 1989. Springer-Verlag. In press.

[Rishe *et al*-89-AM] N. Rishe, D. Tal, and Q. Li. "Architecture for a Massively Parallel Database Machine" *Microprocessing and Microprogramming.* The Euromicro Journal, 1989, in press.

[Rishe/Tal-89-PD] N.Rishe, D.Tal. "Physical database design in the LSDM database machine", in preparation.

[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.

[UNISYS-87] UNISYS Corp. The Database Management System SIM. 1987.