JUNE 1988 VOL. 11 NO. 2

D. Tolbert

a quarterly bulletin of the
Computer Society
technical committee on

Data Engineering

CONTENTS

SPECIAL ISSUE ON WHATEVER HAPPENED TO SEMANTIC MODELING

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS INC
IEEE

THE COMPUTER SOCIETY

SIM: Implementation of a Database Management System Based on a Semantic Data Model

R. L. Guck, B. L. Fritchman,
J. P. Thompson and D. M. Tolbert

Unisys Corporation, Irvine, CA 92718

Abstract

The advantages of semantic data models over hierarchical, network and relational models have been well established. However, a comprehensive database management system (DBMS) that directly supports a semantic data model faces new implementation challenges that stem directly from the model. Some of the more significant issues are:

Data integrity: The DBMS must assume additional data integrity responsibilities due to the semantics of entity types, generalization hierarchies, relationships, and other conceptual-level constraints.

Conceptual object mapping: The conceptual schema must be mapped into lower-level physical components such as records, items, and access methods. However, high-level query parsing and optimization should not require detailed knowledge of mapping techniques so that multiple mapping methods and data sources (i.e. heterogeneous data access) can be adopted more easily.

Performance: Special considerations are needed to address performance due to the greater "distance" between conceptual and physical database objects and the corresponding increase in translation that is required.

The Semantic Information Manager (SIM) is a fully featured, commercially available DBMS implemented for the Unisys A Series line of mainframes. SIM directly supports a semantic data model similar to Hammer and McLeod's SDM [HaMc81]. SIM forms the center of the InfoExec^{tm1} system, which provides a comprehensive data management environment based on the semantic data model. This paper provides a summary of the principal InfoExec modules, and it describes some of the key implementation strategies used by SIM to address the above issues.

1 Semantic Data Models

Hierarchical, network, and relational data models possess inherent limitations that have been identified in many papers [e.g., Kent79, KiMc81, Ship81]. To overcome these limitations, a number of high-level data models have been proposed such as the binary relational model [Abri74], entity-relationship model [Chen76], extended relational model [Codd79], functional data model [Ship81], and the semantic data model [HaMc81]. Each of these models can be termed "semantic" data models because their common objective is to capture more of the meaning of data than previous data models.

1 InfoExec is a trademark of Unisys Corporation.

Precise terminology and concepts vary from one model to the next. However, a core set of semantic data model features can be described as follows:

(1) The primary objects of interest in a semantic data model are "entities" which are classified by "entity types".

(2) Entity types may be "subtypes" or "supertypes" of other entity types, thereby forming "generalization hierarchies". The "root" entity type of a hierarchy is termed a "class", while all other entity types are termed "subclasses".

(3) Entities possess "attributes" which can be "single-valued" or "multi-valued". Attributes establish characteristics of entities including relationships between entities.

(4) Data integrity constraints are expressed in several forms including semantics that are defined for hierarchies, attributes, and relationships.

Many advantages can be realized by a DBMS that is based on a semantic data model. For example, data can be modeled more closely to its real-world perception, thereby simplifying database design and maintenance. The database can assume increased responsibilities for data integrity, thereby reducing application development expense while increasing database reliability and security. A query language can be developed that is conceptually natural, expressive, and easy to use. Also, query optimization can be performed with greater scope.

Overview of SIM and the InfoExec System

The SIM project was initiated five years ago at Unisys with the goal of producing an advanced DBMS for our A Series systems. We chose a semantic data model similar to Hammer and McLeod's SDM [HaMc81] as the basis of SIM for several reasons. Besides offering the advantages described above, SDM is sufficiently powerful to allow a wide variety of existing database systems to be viewed with a single model. At the same time, SDM is sufficiently flexible to allow evolution into even more advanced data modeling concepts (e.g., inference mechanisms could be added to more directly support knowledge-based systems).

SIM offers a formal data definition (DDL) language for schema definition and a data manipulation language (DML) for data access. The DDL supports a rich set of constructs including powerful data types, generalization hierarchies, relationships, and integrity and security constraints. The DML is a high-level, non-procedural language that supports powerful features while addressing ease of use via an English-like syntax. These topics are fully discussed in [JFGT88].

SIM is the central component of the InfoExec system, which provides a complete data management environment based on the semantic data model[2]. The InfoExec environment addresses global data management issues, such as

[2] The term "DBMS" implies the software which is used to define, access and maintain a database. A "data management environment" provides additional software, such as a data dictionary and ad-hoc query products, to assist the overall application development effort.

productivity, ease of use, and performance, by providing additional modules which address the development and use of both the database and its applications. In contrast to modeling or translation facilities, experimental prototypes, and hybrid implementations [e.g., AnHa87, GGKZ85, MBW80, PSM87, Ship81, TsZa84], the InfoExec system represents a "ground-up" implementation that propagates the semantic data model throughout its interfaces. We believe that ours is the first fully featured, commercially available system based on a semantic data model.

A brief description of the InfoExec modules which are integrated with SIM is provided below.

Data Dictionary

A key module of the InfoExec system is the Advanced Data Dictionary System (ADDS) which is used to define, maintain, track, and perform other functions for all SIM databases (hence, ADDS is an "active" dictionary). ADDS also provides many other definition capabilities which support the application environment. For example, ADDS can be used to define record structures, screen descriptions, program data structures, and documentation. All of the ADDS functions are offered via a menu-oriented interface which minimizes the need to know specific sublanguages (i.e., a SIM database can be defined without having to know the SIM DDL). Each menu offers extensive on-line "help" text to reduce the need to access reference manuals. ADDS is implemented as a fairly large SIM database application, thereby making use of (and giving credence to) the power of the semantic data model.

Ad-hoc Query Facilities

Two ad-hoc query facilities are provided to support on-line access to the database. The Interactive Query Facility (IQF) offers a screen-based interface via terminals while the Workstation Query Facility (WQF) offers a graphically-oriented interface via intelligent workstations. Both products preserve the data model and its expressive abilities, yet they reduce the need for the end-user to directly use the underlying DML by using prompts or graphic icons to construct query statements. Both products have dictionary interfaces for saving and reusing queries, and both products provide extensive report handling features.

Host Language Interfaces

Host language interfaces to SIM databases and ADDS are provided in COBOL, Pascal, and ALGOL. The interfaces are unique in that they provide the SIM DML as language extensions "tuned" to the flavor of each host language. This approach preserves the full expressive power of the SIM DML while allowing query statements to be intermixed with host language constructs. For example, queries may contain expressions that reference program variables and functions, and queries may be passed as parameters within and between programs. ADDS can be accessed for importing conceptual schema elements, file descriptions, data structures, screen interfaces, and other definitions.

2.4 Data Management System II (DMSII)

DMSII is a multi-user, high-performance, network-based DBMS that has been widely-used on A Series systems for nearly 15 years. DMSII provides a complete set of database features in the areas of file structures and access methods, resource management, and auditing and recovery mechanisms. SIM exploits these features by using DMSII as its primary physical access engine (although access to other data sources is also supported). This arrangement results in an integrated co-existence between DMSII and SIM. For example, an InfoExec utility is provided to view an existing DMSII database as a SIM database.

Other Database Utilities

The InfoExec system also offers a complete range of database support utilities for performance monitoring and tuning, backup and recovery, and other functions. Due to the integrated relationship of DMSII and SIM, most of these utilities will operate on either DMSII or SIM databases. A screen-oriented interface to these facilities is provided via the Operations Control Manager (OCM) to eliminate the need for learning sublanguages peculiar to each utility. The screen interface uses the same on-line help mechanisms used by other screen-based InfoExec products.

3. Implementation Strategy of SIM

Process Architecture Overview

The key implementation objectives of SIM are to (1) realize and directly support the data model, (2) provide flexible conceptual object mapping with useful alternatives, (3) provide query (DML) parsing and optimization that is independent of specific mapping techniques, (4) enforce the data integrity constraints expressed by the database schema, and (5) provide query execution performance that is suitable for databases requiring high transaction rates. Two longer-term goals are to provide access to both heterogeneous and distributed data.

As a consequence of these criteria, the SIM architecture is designed to be highly modular yet efficient. This architecture is depicted in Figure 1. As shown, SIM is composed of multiple layers that are joined at distinct interfaces. Each software module operating within each layer has a specific perspective of the database and performs its operations based on that perspective. This approach allows each module to function at the highest possible level, thereby minimizing its need to know details of lower-level modules. A key performance strategy of this architecture is the use of dynamic code generation for query execution. Also, the implementation of SIM exploits architectural advantages of A Series systems (the discussion of which is beyond the scope of this paper).

An important implementation strategy, the Logical Underlying Component (LUC) model, is described in the next section. The purpose and operation of each software module within SIM's architecture are described in subsequent sections.

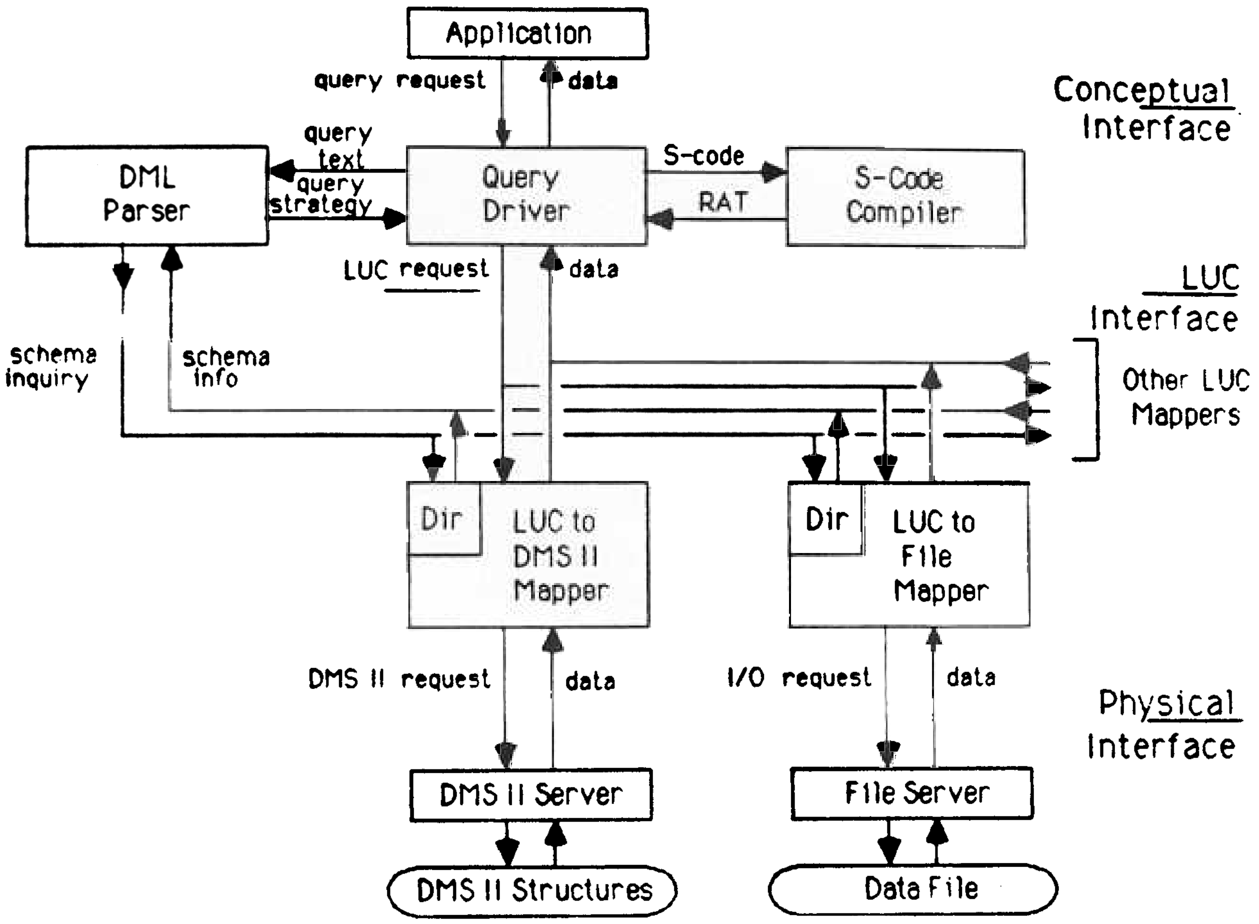

Figure 1 - SIM Architecture

3.2 The LUC Model

Because we wish to separate conceptual-level software modules from physical mapping details as much as possible, we have introduced an intermediate data model within SIM's process architecture. It is termed the "LUC" model because it consists of "logical underlying components" which are functionally simpler than the conceptual-level objects in SIM. There are three types of LUC objects:

(1 A "record" LUC consists of a fixed set of single-valued fields. Each field has a fixed size, location, and a known data type.

(2) A "relationship" LUC defines a unidirectional relationship between an "owner" record LUC and a "target" record LUC. The target record LUC may be "dependent" which means that it cannot exist without the owner LUC object. Two relationship LUCs may be identified as "inverses" of each other.

(3) An "index" LUC defines an access method on a record LUC. The index may have multiple ascending and/or descending keys, and it may be a random or index-sequential structure.

Each LUC object has a well-defined set of operations that can be performed on it and a corresponding set of integrity rules. Every conceptual schema has a precise mapping into a LUC schema that is independent of how the LUC schema is mapped into the underlying physical schema. Furthermore, the LUC model is sufficiently flexible that it can be mapped into virtually any record-based model, including conventional database systems. Thus, the software "above" the LUC level need only know how to translate conceptual-level operations into LUC level operations, and software "below" the LUC level need only know how to translate these operations into corresponding operations on the particular data source(s) involved.

Application

A SIM application is either an ad-hoc query facility such as IQF or WQF, a host language compiler such as COBOL or Pascal, or a user-written host language program. In each case, the application's view of the database is based entirely on the database's conceptual schema. Each application submits separate requests to parse and execute DML queries, thus allowing the two processes to occur independently. For example, a host language program's queries are parsed at compile-time but executed at run-time. Once a query is parsed, it can be executed multiple times, and multiple, simultaneous invocations (e.g., with different input parameters) are allowed.

3.4 Query Driver

The Query Driver is the application interface to SIM. Its responsibility is to manage the query parsing and execution requests submitted by the application program. An application requests that a query be parsed by submitting a DML query as a text string. The Query Driver submits this text to the DML Parser, which returns a query execution strategy, one part of which is an "S-code" (semi-compiled code) symbolic program. The Query Driver submits the S-code source to the S-code compiler and in return receives a block of machine code called a remap-and-test (RAT) procedure.

An application executes a query by submitting "retrieve data" or "process update" requests. The Query Driver responds to these requests by interpretively executing portions of the query's execution strategy. This causes LUC requests (including the appropriate RAT procedure) to be submitted to the appropriate LUC Mappers. The LUC Mappers respond to retrieve requests with data obtained from the database or with update results sent to the database.

DML Parser

The DML Parser parses, validates, and develops an optimized execution strategy for each DML query. While parsing a query, the DML Parser retrieves schema information by submitting schema inquiry requests to the appropriate LUC Mappers. The parser receives information on conceptual and LUC objects and the mapping between them. Information such as relationship cardinalities, available indexes, and integrity constraints is used to develop an optimal execution strategy for the query.

The information received by the DML Parser is free from physical mapping details except for the layout of LUC records. For each LUC record, the parser obtains such details as field offsets, lengths, and data types. This information is used to generate the S-code program, which is returned as part of the execution strategy.

Many integrity constraints are the responsibility of the DML Parser. For example, data type, range, and existence requirements are either verified during parsing, or the query's execution strategy is augmented such that enforcement is performed during query execution. Other integrity constraints are the responsibility of the LUC Mapper.

3.6 S-code Compiler

The S-code Compiler is a dynamic code generator. The S-code language consists of a limited set of statements, operators, and operands. A typical S-code "source" program consists of a few hundred bytes, and the S-code compiler converts this program into the RAT procedure previously mentioned. When called, a RAT procedure accepts a set of buffers (arrays) and performs testing and update operations on them as instructed by its source program. The RAT performs two basic kinds of operations which contribute greatly to the performance of query execution:

(1) The RAT can examine a record and apply a selection criterion to determine if the record should be selected. This "test" function is used to find database records during search phases of query execution.

(2) The RAT can move data from one record to another, performing expression analysis (including field-level integrity checks), data type conversion, and other functions. This "remap" function is used to map data between database records and application data structures.

Typically, the S-code compiler can compile an S-code source program into a RAT in less than one second. Since the RAT procedures perform all expression analysis and data transfer for a query, the performance impact of the interpretive nature of the remaining SIM modules is minimized.

3.7 LUC Mappers

A different LUC Mapper exists for each type of physical data server that can be accessed as part of a SIM database. Each LUC Mapper accepts LUC commands and translates them into appropriate calls on the underlying data server module. The Mapper calls the RAT procedure to filter and/or update records retrieved from the data server. The Mapper exploits any functionality that is available from the data server (e.g., transaction control, locking mechanisms, searching mechanisms, etc.). When necessary, the Mapper supplements the data server's functionality with anything that is lacking.

The LUC Mapper plays an important part in enforcing data integrity during query execution. Many of the constraints implied by the data model or the conceptual schema are automatically enforced by the Mapper. For example, existence and cardinality constraints on relationships are enforced by the Mapper during update operations. Dependent LUC objects which cannot exist without a corresponding owner are either automatically deleted when the owner is deleted, or the Mapper disallows the deletion of the owner (or the relationship to the owner) depending upon the type of conceptual object represented. For example, when an entity is deleted from a subclass role, all relationships and multi-valued attributes are deleted from both the subclass and all subordinate roles in which the entity participates. These operations are the responsibility of the LUC

Mapper so that it may exploit any physical mapping properties for greater performance.

Each LUC Mapper also services schema inquiry requests from the DML Parser. A portion of the LUC Mapper called the Directory is dedicated to this function. Directory information is stored either in the data dictionary (ADDS) or within the database itself.

Note that the use of multiple LUC Mappers allows a flexible composition and distribution of a single database. What is viewed as a single SIM database at the conceptual level can consist of data that is derived from multiple data sources. A single data server may distribute data within its own control, or data can be distributed by one or more LUC Mappers. Since the LUC Mapper's interfaces are message-oriented, it can reside on a foreign host, thereby allowing distributed data. This architecture also allows a single query to access data within multiple databases.

3.8 Physical Data Servers

Ultimately, SIM aims to allow any data source to be a data server, thereby allowing virtually anything to be accessed as a SIM database. Initially, however, the primary physical data server for SIM databases is provided by DMSII, which, as mentioned earlier, is a robust and widely-used network-based DBMS. Also, a File Mapper is available which provides access to simple data files, although only inquiry access is currently allowed.

To further enhance the performance of DMSII-mapped databases, the DMSII data server has been extended to accept and use RAT procedures during record selection. When this is done, a RAT is accessing low-level I/O buffers directly, thereby eliminating a great deal of potential data movement. Also, the RAT can search multiple records sequentially until a record which meets the selection criteria is found. This prevents returning from DMSII with unneeded records. Finally, the only records that are locked during updates are those selected by the RAT, hence, the locking of non-essential records is eliminated.

4. Conceptual Object Mapping

SIM offers multiple techniques for mapping conceptual objects into physical DMSII components. Default mapping algorithms are chosen to provide a careful balance between performance and memory/disk usage. Alternatives are available so that users may "tune" the mapping to conditions present for a specific database or host system. The default mapping techniques and some of the alternatives are described below.

Classes: Each class is mapped to a separate disjoint structure containing fixed-format records. Each record holds data items corresponding to the class's attributes plus a system-generated surrogate item. The surrogate item contains a value that gives each entity a unique identity that never changes over the entity's life. The class is spanned by a surrogate index, which is an access method whose key is the surrogate item. This provides efficient entity retrieval by surrogate value. User alternatives include structure types which employ different storage and access techniques and different surrogate implementations including the use of a suitable required, unique attribute.

Subclasses: Subclasses have two possible mappings depending upon their superclass configurations. When a subclass has a single immediate superclass and all of its sibling subclasses are mutually exclusive, the subclass is mapped to a variable-format extension of its superclass's records. A subclass with multiple superclasses and/or non-mutually exclusive siblings is mapped to a new disjoint structure, and a "copy" surrogate item and index are added to that structure. The copy surrogate item contains the same surrogate value as the entity's base class. The user may choose the disjoint structure mapping when the variable format mapping would have been the default, and he or she has the same structure type options that are available for classes.

Attributes: Attributes whose range is a system-defined class representing a printable data type (such as arithmetics, strings, and booleans) are termed "data-valued attributes" (DVAs). All other attributes range over some user-defined class and are termed "entity-valued attributes" (EVAs). SIM employs different mapping techniques for single-valued (SV) and multi-valued (MV) DVAs and EVAs.

Each SV DVA is mapped into two items within its owner's record: a data item holds the value of the DVA, and a "null bit" item indicates if the value exists. An MV DVA is normally mapped to a disjoint structure containing a data item, copy surrogate item, and surrogate index similar to that used for subclasses. When the DVA meets certain requirements, it can be mapped to an embedded structure or an array contained in the structure of the DVA owner.

Each EVA has an "inverse" EVA which is either explicitly or implicitly declared. An EVA pair forms a bidirectional relationship that is mapped with the cardinality of each EVA taken into consideration. Each EVA of a 1:1 relationship is mapped into an item within its owner's record. The item contains the surrogate value of the owner of the inverse EVA. As an alternative, the user can choose absolute address pointers for each EVA of a 1:1 relationship.

By default, both EVAs of 1:many and many:many relationships are mapped to a structure called the "common EVA structure." Each record represents a single relationship instance between two entities. Each record possesses two items containing the surrogate values of the entities plus a relationship type item. Many relationships are mapped to a single common EVA structure, thereby minimizing the number of structures consumed by the database. As alternatives, the user may choose from several absolute address and foreign key mappings to achieve better performance at a cost of an increase in the number of structures used.

5. Conclusions

Key implementation strategies have been described that are used by SIM, a DBMS based on a semantic data model. The SIM architecture is highly modular and employs an intermediate data model called the LUC model to increase the independence of the conceptual and physical layers. As a result, query parsing and optimization do not require detailed physical mapping knowledge, yet the system allows multiple physical mapping alternatives and distributed, heterogeneous data access. Data integrity is enforced by a combination of query augmentation and dynamic constraint enforcement performed by the LUC Mapper modules. Query performance techniques include high-level query optimization, dynamic code generation, and LUC Mapper exploitation of data server features.

Our experience thus far with SIM has been very satisfying. Our tests have demonstrated performance comparable with conventional database systems, and our initial customers have testified to the power and productivity provided by the model. Investigations for future work include DDL and DML enhancements, including additional integrity mechanisms, further performance improvements, on-line reorganization, increased access to distributed and heterogeneous data access, and temporal data.

References

[Abri74] J. R. Abrial, Data Semantics. In Database Management, J. Klimbie and K. Koffeman Eds. North-Holland Amsterdam, 1974.

[AnHa87] T. Andrews and C. Harris, Combining Language and Database Advances in an Object-Oriented Development Environment, In proceedings of ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, 4-8, October 1987

[Chen76] P.P.S. Chen, The entity-relationship Model: Towards a unified view of data, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976

[Codd79] E. F. Codd, Extending the Database Relational Model to capture more meaning, ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979

[GGKZ85] K. Goldman, S. Goldman, P. Kanellakis, and S. Zdonik, ISIS: Interface for a Semantic Information System, In proceedings of ACM SIGMOD International Conference on Management of Data, May 1985

[HaMc81] M. Hammer and D. McLeod, Database Description with SDM: A Semantic Data Model, ACM Transactions on Database Systems, Vol. 6, No. 3, September 1981

[JFGT88] D. Jagannathan, B. L. Fritchman, R. L. Guck, J. P. Thompson, and D. M. Tolbert, SIM: A Database System Based on the Semantic Data Model, To appear in proceedings of ACM SIGMOD International Conference on Management of Data, 1988

[Kent79] W. Kent, Limitations of Record-Based Information Models, ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979

[KiMc81] R. King and D. McLeod, Semantic Data Models, in S. B. Yao (Ed). Principles of Database Design, Prentice Hall, 1984

[MBW80] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong, A Language Facility for Designing Database-Intensive Applications, ACM Transactions on Database Systems, Vol. 5, No. 2, June 1980

[PSM87] A. Purdy, B. Schuchardt, and D. Maier, Integrating an Object Server with Other Worlds, ACM Transactions on Office Information Systems, Vol. 5, No. 1, January, 1987

[Ship81] D. W. Shipman, The Functional Data Model and the Data Language DAPLEX, ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981

[TsZa84] S. Tsur and C. Zaniolo, In Implementation of GEM - supporting a semantic data model on a relational back-end, In proceedings of ACM SIGMOD International Conference on Management of Data, May 1984