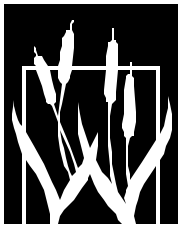


Whitemarsh
Information Systems Corporation

Great News,
The Relational Data Model is Dead !

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: mmgorman@wiscorp.com
Web: www.wiscorp.com

Table of Contents

1. Introduction

Great news, the relational data model is dead! Well, not completely. It's that the relational data model, as we all know it through its linguistic expression, the SQL language, has been "dramatically extended" by the ANSI H2 Technical Committee on Database. The most interesting part of this "dramatic extension" into what is now called SQL:1999, is that it has taken the SQL data model clearly into the past, and then beyond. That is, to embrace the data model of a production class of database management systems (DBMSs) that predate the first "discovery" of the relational data model. To understand why these "dramatic extensions" take the SQL data model clearly into the past, this paper presents:

- An overview of data models,
- The four data models and the identification of the DBMSs that adhere to these data models,
- The SQL language,
- Data models and the SQL language,

,

- The SQL:1999 language,
- SQL:1999's impacts on the relational data model, and
- SQL:1999's impacts of SQL:1999 on database applications.

2. What Is a Data Model

A data model consists of three main components:

- Record Structures
- Relationships
- Operations

2.1 Record Structures

A record structure (in relational terms, a table) represents a set of data fields (in relational terms, a column). There are several classes of data fields. These include:

- Single Value–where each field represents a single value such as Birthdate with the value 11/11/1987
- Multi-value–where each field represents multiple values such as Nicknames with values "Buddy, Guy, Mac"
- Groups–where each field has subfields to represent single-set of values such as Address with Street-1, Street-2, City, State, Zip
- Repeating Groups–where each field has subfields to represent multi-sets of values such as Dependents that contains subfields, Dependent Name, Dependent Birth date, Dependent SSN.
- Nested Repeating Groups–where each field has subfields to represent multi-sets of values such as Hobby (Hobby Name, Hobby Description, Hobby Annual Cost) within each multi-set field, Dependent, that also contains single value fields such as Dependent Name, Dependent Birth Date, Dependent SSN that is within the Employee table that also contains single-valued fields, Employee SSN, Employee birth-date, etc.

2.2 Relationships

The second part of a data model is relationships. Relationships are explicit linguistic expressions that defines the basis for interrelating records from different types of record structures. Relationships among values of multi-valued fields, instances of repeating groups, and instances of nested repeating groups within record structures are implicit and are completely controlled by the DBMS. The eight types of relationships that manage the relationships between records from different record structures are:

- One-to-one: Two record structures representing the partitioning of a single record structure into two parts, for example, dividing a record structure of 400 columns into two 200 column sets.

- One-to-many (single member): One record structure with one instance and another record structure with multiple instances, for example, employee and job skills

- One-to-many (multiple members): One record structure with one instance and more than one other record structures, each with multiple instances, for example, territory and then salesman and customers.

- Many-to-many: One record structure with multiple instances related another record structure with multiple instances. For example, the relationship that would exist between multiple owners of multiple automobiles.

- Singular, single member: One record structure with multiple instances. For example, top rated employees

- Singular, multiple member: Multiple record structures with multiple instances. For example, current employees, high achievers, full-time employees, part-time employees, and retired-employees.

- Inferential: Multiple record structures with multiple instances such that the relationship is not explicitly based on shared values. For example, an employee is assigned to a department, and four buildings are designated as containing employees of that department. The inferential relationship is the one between the employee and the buildings that infers that an employee may be in one or more buildings, but does not explicitly identify which building.

- Recursive: One record structure with multiple instances that are interrelated hierarchically. For example, organization unit which contains organization units, each of which contain organization units, etc.

The relationship mechanisms exist either 1) as separately defined and controlled DBMS "pointer" mechanisms, or 2) as shared data field values that reside within different record structures that are defined through normal database design techniques and whose data values are completely controlled through end-user application program logic.

2.2.1 DBMS Defined and Controlled Relationships

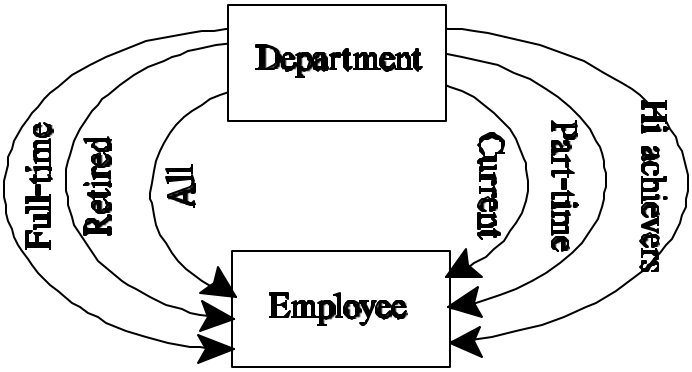


Figure 1. Multiple Relationships between Department and Employee

Relationship mechanisms are generally stored by DBMSs in one of two places: In the "source" (also called the "owner") data record of the relationship (or possibly in a separate structure just for relationships), or in the "target" (also called the "member") data record of the relationship. In this paper the terms "owner" and "member" are employed.

Many DBMSs that support DBMS defined and controlled relationship processing, maintain four types of relationship pointers: owner, member, next, and prior. The owner pointer enables access of the owner data record of a relationship. The member relationship mechanism enables an owner record to access the first member of a given relationship. Member data records contain "prior" and "next"relationship mechanisms in a given data record with respect to either the "prior" or the "next" record in a set of records that form a given relationship. Finally, if member data records are ordered with respect to the owner data record then that order is maintained via member relationship mechanisms.

For example, there may be an overarching relationship between a department and its employees. There then may also be other relationships for current, full-time, hi-achievers, part time and retired employees. Figure 1 shows the record structures and these relationships between them.

Most of the DBMSs that support pointer-based relationships permit any reasonable quantity of different relationships between a given owner and sets of members. Some DBMSs store all member relationship mechanisms of a given owner-member relationship in the owner record. This technique supports quick counts of members for a given owner and also quick access to a specific member.

2.2.2 Shared Data Value Relationships

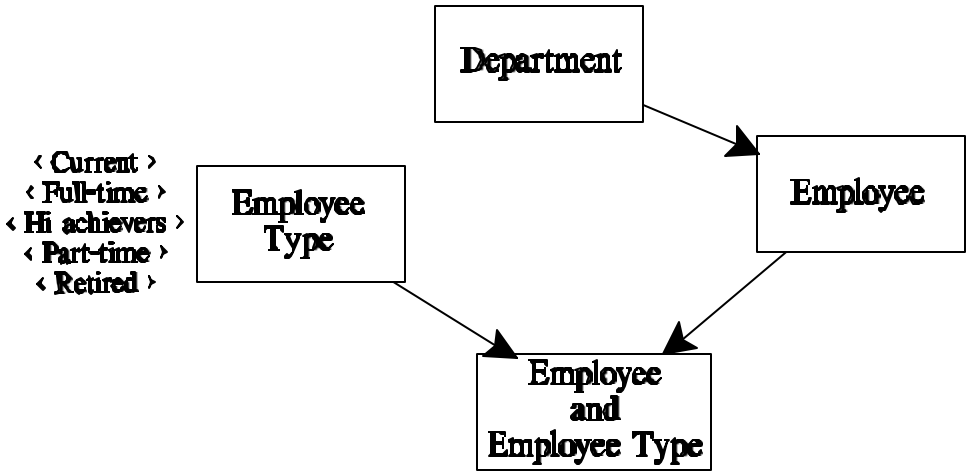


Figure 2. Relational tables necessary for multiple relationships between Department and Employee

The second form of relationships, shared data values, most often takes on the form of a shared data value that is the primary key value of the owner data record stored as a foreign key value in one or more member data records. This member data field is called a foreign key because it is properly a field from the owner data record.

When the owner data record is needed, the foreign key's data value is used by the DBMS to select the owner data record. When all the member data records related to a specific owner data record are desired, the foreign key value is used to select all the member data records. These accesses are most commonly accomplished through the use of indexes.

If multiple relationships are needed between an owner record and a set of members then there must be a different field within the member data record to represent each relationship. For example, there would have to be different fields for current, full-time, hi-achievers, part time, and retired employees. Such a technique generally makes for bad database design, however. To make a "good" database design, several more database record structures have to be created. Figure 2 shows the four different record structures that have to be defined to create these relationships.

2.2.2 Comparative Performance

DBMSs that employ DBMS defined and controlled relationship mechanisms that store member pointer arrays in the owner record, or that store next, prior, and owner pointers in the member data records usually retrieve and traverse data much faster than those DBMSs that only support shared field value relationship processing mechanisms that are based on primary and foreign keys.

The DBMS defined and controlled relationship mechanism DBMSs, however, take longer to load and update data records because in addition to updating data records, their relationship mechanisms have to be updated as well. These DBMSs also take longer to accomplish database reorganizations when relationships between data records have to change.

In contrast, the shared-value relationship DBMSs can load and update data records faster as all relationship processing (save enforced referential integrity) is deferred until retrieval. Retrieval processing is generally slower as significant computer processing is expended to reassemble report records that exist in the database within separate normalized tables.

2.3 Operations

The third part of a data model is operations. Here, there are two types: Record structure operations and relationship operations. Record operations include the traditional ones of Add, Delete, and Modify. Relationship operations are of two classes and depend directly on whether the relationship mechanisms are DBMS defined and controlled, that is, member pointer arrays stored in the owner record, or are next, prior, and owner pointers stored in the member data records, or are shared value-based, that is, primary and foreign keys.

The choice of relationship implementation gives rise to two different styles of operations: record-at-a-time, or a multi-set of selected records. Record-at-a-time implies that the user's program navigates through the database and employs operations such as GET OWNER, GET NEXT, and GET MEMBER to accomplish the traversal. The set relationship operations are founded mainly on mathematical set operations upon selected sets of records. Included are PROJECT, DIVIDE, JOIN INTERSECTION and UNION. As DBMSs implemented these set processing operations they provides either explicit syntax or syntax support.

2.4 Data Model Summary

These three data model components, record structure, relationships, and operations, in combination, form a DBMS's data model. Different DBMS vendors, over the years have employed different combinations of techniques from these three data model components to form a particular style. These styles, when coupled with the vendor's special language for data structure definition, data loading, query, report writing, and other features give rise to particular DBMSs.

Until ANSI standardized the SQL language, DBMS vendors had no stable and non-proprietary target, nor incentive to either directly implement the facilities implied by the standard SQL language or to map their DBMS facilities to those needed by the SQL language. Within about 5 years after the standardization of the SQL language, DBMS vendors recognized that while they may not have had

explicit implementations, they could create mappings to DBMS engine facilities such that users could employ the SQL language to both define and access databases. This gave those vendors a real advantage as they could both support their legacy DBMS applications and also those applications that were defined exclusively through the ANSI standard SQL language.

3. The Four Data Models

DBMSs from the various vendors have implemented different combinations of characteristics from the data models presented in Section 2. From the late 1960s through the late 1980s, there was an attempt to standardize DBMSs that conformed to the network data model through the organization called CODASYL (Committee on Data Systems and Languages). The documents produced by the CODASYL DDLC (data definition language committee) were called Journals of Development. They were not, however, ANSI standards. Thus, vendors adhered to these JODs "spiritually."

DBMSs that conform at least "spiritually" to the CODASYL JODs are called CODASYL DBMSs. The relationships shown in Figure 1, for example, hi-achievers are called CODASYL sets. The relationships supported by CODASYL DBMSs are:

- One-to-one
- One-to-many (single member)
- One-to-many (multiple members).
- Singular, single member
- Singular, multiple member

The member data records belonging to these relationships are able to ordered and these orderings are automatically maintained by the DBMS.

Because there were neither ANSI standards nor conformance tests during this 1955-1970 time-frame to judge adherence to standards, every DBMS was somewhat different. Notwithstanding, four general data models arose. Figure 3 presents the key characteristics of the four data models. This figure shows the four data models as the columns and the three main characteristics, that is, record structures, relationships, and operations as the rows of this table. Figure 4 presents the lexicon for the abbreviations in Figure 3.

During the first 15 years of DBMS (1955 through 1970) the three data models were:

- Network
- Hierarchical
- Independent logical file

The network and hierarchical data models are characterized by complex record structures, that contain:

- Single Value
- Multi-value
- Groups
- Repeating Groups

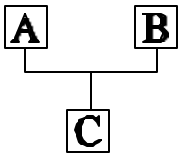
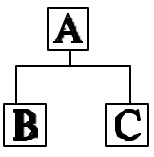
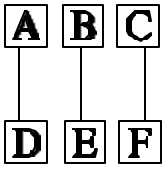
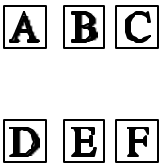
		Data Model			
Characteristics		**Network**	**Hierarchic**	**Independant Logical File**	**Relational**
Record Organization		SV, MV, MD, G, RG	SV segments	SV, MV, RG	SV
Relationship (REL)					
Operations (OPS)	**Record**	A, D, F, M	A, D, F, M	A, D, F, M	A, D, F, M, P
	Relation-ships	C, D, GO, GM, GN	GO, GM, GN	J, INT DIV, UN DIF	J, INT, DIV, UN PR, DIV
DDL		REL, RO	REL, RO	RO	RO
DML		OPS	OPS	REL & OPS	REL & OPS

Figure 3. The four data models most commonly present in DBMSs.

- Nested Repeating Groups

Record Organization	SV = Single Value		
	MV = Multiple Value		
	MD = Multiple Dimension		
	G = Group		
Operations	Record	A = Add	
		D = Delete	
		M = Modify	
		P = Project	
		F = Find	
	Relationship	C = Connect	DIS = Disconnect
		P = Project	J = Join
		DIV = Divide	GO = Get Owner
		GM = Get Member	GN = Get Next
		INT = Intersection	PR = Product
		UN = Union	DIF = Difference

Figure 4. Lexicon for Figure 3

Network data model DBMSs generally support explicit relationships between records structures such that a member data record can be related to more than one owner data record. In contrast, a hierarchical data model DBMS only allows a relationship between one or more members and a owner data record. IBM's IMS created facilities to simulate networks through a concept called "logical twins."

In all the network and hierarchical DBMSs, the relationship mechanisms between data records are through DBMS generated and controlled pointers.

Independent logical file data model DBMSs support value-based primary and foreign key relationships between data records. The record structures allow the definition of naturally existing hierarchical data structures. In addition, some independent logical file DBMSs permit outer references from within a record structure to other record structures within the same database. This enables both many-to-many and inferential relationships.

Once network and hierarchical database designs are stabilized, processing is very fast. While relationship processing between data records is slower for independent logical file DBMSs, their database designs are more flexible and can be redesigned more easily and quickly.

The relational data model was proposed in 1970 and the first relational DBMSs entered production status by the end of the 1970s. Today, relational DBMSs such as Oracle, Sybase, Informix, DB/2 and SQL Server are the most commonly found DBMSs in business, government and industry. As proposed in 1970, relational data model DBMSs only support single valued fields and relationships between tables are shared data value-based primary and foreign keys.

Relational DBMSs are the most flexible of all the four data model DBMSs with respect to three features:

- The ability to change relationships quickly and easily. For example, a new data record can be added and related to existing data records through shared data values. There is no massive database reorganization effort.

- The ability to hypothesize previously unknown relationships. For example, if there is a need to relate the persons with an MA degree with students born in MA (i.e., Massachusetts), the query language directly supports the expression of the relationship which is then processed.

- The ability to evolve database designs from a few data records to many data records in an incremental way.

Once a relational database's design is mature, however, the very features that make it the most flexible also makes it the slowest when compared to DBMSs conforming to the other three data models. To compensate for this lack of speed, relational DBMS vendors have, over the years, created for example, very sophisticated index mechanisms, buffer management techniques, and data record clustering to physically group data records from different types together.

Figure 5 presents an enumeration of DBMSs by data models. In this figure, the dates relate to the earliest appearance of one or more of the DBMSs in a production status with one or more clients.

The main differences among data models are in the area of record structures and relationships. In the network, hierarchical, and independent logical file data models, data that naturally exists within nested structures can be explicitly defined. In contrast, in the relational data model, each record structure can only be a set of single valued fields. Thus, if there are naturally existing nested data structures then for each,

- A separate record structure must be defined
- A complete primary key must be specified
- Foreign keys for each relationship must be specified

Some relational data model DBMSs maintain each record structure as a separate physical file with all the associated computer resource overheads. Other relational data model DBMSs permit one or more of the separate record structures to be contained within a single physical file. While relational purists may assert that there is some great theoretic need for all this extra work and overhead, there are certainly no database or application compelling reasons to completely separate naturally existing data hierarchies into distinct relational tables. The cost of doing so is manifest in the extra computers, application development and human resources necessary to create, process, and maintain them, and these extra costs do not compare favorably with the derived benefits.

	Data Models			
	Network (mid 1960s)	**Hierarchy (early 1960s)**	**Independent Logical File (late 1960s)**	**Relational (late 1970s)**
Typical systems	Supra IDMS/R IDS DMS-2 DMS-1100 VAX/DBMS	System 2000 IMS	Inquire Adabas GIM family Nomad Focus Ramis CA/Datacom Model 204	DB/2 CA/ingress Oracle Sybase Informix SQL/server
Governing ANSI standard	ANSI-NDL		ANSI-SQL:1999	ANSI-SQL/86 ANSI-SQL/89 ANSI-SQL/92

Figure 5. DBMSs by Data Model and Governing ANSI Standard

4. The SQL Language

The SQL language, originally known as the structured query language, was developed to support the relational data model. The language was created by IBM in the early 1970s. Ultimately, the language was made public domain by IBM and was then standardized by the ANSI NCITS (National Committee for Information Technology Standards) H2 Committee on Database in 1986. Note: H2 stands for nothing, it's just the committee's "primary key." The SQL language consists of the following main components:

- Database and data record structure definition including relationship integrity specification, and views.

- Data record operations for insert, update and delete, and relationship operations that accomplish JOIN, PROJECT, DIVIDE, INTERSECTION, and DIFFERENCE.

- Data record selection operations from a single data record or through nested subqueries to then select shared data value related data records.

- Privacy definition and control

- Concurrent update and retrieval data control

- Transaction processing

Because the SQL language is commonly employed through traditional programming languages like COBOL, SQL contains record-at-a-time processing commands, that is, cursor operations, that operate against selected sets of records.

The SQL language was first standardized in 1986. The basis for forming the standard was to standardize only those facilities that the vendor-members of the H2 committee could agree upon. Simply, the goal of the standard was to fix a base-line of established practice. From 1986 through to the next standard, 1989, other features were standardized including referential integrity. Referential integrity is an old concept and has been in CODASYL network systems since the late 1960s.

The next SQL standard was brought out in 1992. This was a major upgrade to the SQL language. The extensions were mainly in integrity constraints, multiple-language support, transaction processing, full referential integrity, and the like. The fundamental components of the underlying data structures and relationship processing remained the same. That is, tables that contain only single-valued fields.

Figure 6 presents a table that shows the incremental features above the basic capabilities contained in SQL/86

5. Data Models and the SQL Language

By the time the SQL language became standardized by ANSI in 1986, several vendors, such as IBM (DB/2), Oracle, Sybase, and Informix had become very popular. These DBMSs gained market share against non-SQL DBMSs such as Cullinet's IDMS, IBM's IMS, Information Builder's Focus, and Software Ag's Adabas. The non-SQL DBMS vendors were then under significant pressure to either develop SQL language interfaces to their systems or completely transform their DBMSs to the relational data model. In the next 10 years, these vendors created SQL interfaces. By the end of the 1980s, it had become very clear that database designers, programmers, and end users could employ the SQL language to accomplish their activities without having a DBMS that was built on a strictly relational database engine.

SQL/1986	SQL/1989	SQL/1992
Basic features, that is	SQL/1986 plus	SQL/1989 plus
Tables Columns Views Basic relational operations Some integrity constraints Language bindings to COBOL, FORTRAN, C, etc.	Partial Referential Integrity	Assertions Bit data type CASE Character Sets Connection Management DATETIME Domains Dynamic SQL Enhanced constraints Full Referential Integrity Get Diagnostics Grouped operations Information Schema Multiple module support National character sets Natural joins (inner & outer) Row & Table constraints Schema manipulation Subqueries in check clauses Table constraints Temporary tables Transaction Management Union and intersect

Figure 6. Features of SQL/86, 89 and 92

6. SQL:1999 Language

Starting in 1992, the H2 committee began the development of dramatic extensions to the SQL/92 standard. The greatest change in the standard is that it no longer adheres to the 1970 relational data model. The second biggest change is that the SQL:1999 language now consists of individual parts that comprise a foundation and then a series of independently specified packages. The remainder of this section provides an overview, in outline form, of the contents of the SQL:1999 standard.

6.1 Foundation Components

- Tables that have been enhanced to support new built-in data types (boolean, enumerated, extensions to character sets, translations, and collations)

- BLOB and CLOB data types

- Abstract Data Types (user defined data type with behavior, an encapsulated internal structure, and access characteristics of public, protected, or private)
  - strong typing
  - subtypes and inheritance
  - encapsulation
  - virtual attributes
  - substitutability
  - polymorphic routines
  - dynamic binding
  - compile time type checking
  - value ADTs

- Array

- Row Types (table person (SSN, name(first, middle, last), address(street, city, state, zip(four, five)))))

- User Defined Functions

- Predicate extensions (for all, for some, similar to, cursor extensions, null values, assertions, view updatability, joins)

- Triggers
  - Different triggering events, update, delete, and insert
  - Optional condition
  - Activation time: before and after
  - Multiple statement action
  - Several triggers per table
  - User-defined ordering
  - Condition and multiple statement action per each row or per statement

- Roles (enhancements to security), & Savepoints
- Recursion

6.2 Call Level Interface

The SQL Call Level Interface is the set of language specifications used by DBMS vendors to enable direct SQL engine access through completely specified call routines. Microsoft, for example has implemented SQL/CLI and calls it ODBC.

The CLI specification contains more than 50 different call specifications that address:

- Connection control to SQL servers
- Allocate and de-allocate resource
- Execute SQL statements
- Obtain diagnostic information
- Control transaction termination
- Obtain information about implementation

It also contains Resource Management Handle routines for:

- Environment
- Connection
- Statement
- Context

The CLI also contains a Descriptor Area that accommodates:

- Application parameter
- Application row
- Implementation parameter
- Implementation row

6.3 SQL/Multi Media (MM) Components

SQL/MM is itself a set of subparts that contain full specifications for a discrete set of data management functionality to address the data processing needs of:

- Full Text
- Spatial
- General Purpose
- Still Image

6.4 SQL Persistent Stored Module Language Components

SQL:1999 now has a complete embedded programming language to support the processing needs of its user defined data types, assertions, and triggers. The SQL/PSM language supports the following capabilities:

- Call
- Return
- Compound Statements (Begin ... End)
- If Statements
- Case Statements
- Loop
- Repeat
- While
- For
- Leave
- Assignment
- Signal and Resignal

6.5 SQL Transaction and Connection Management

Since the advent of distributed processing, client-server, and of course the Internet, the ability to manage transactions is critical. SQL:1999 now has facilities that address the following areas of transaction and connection management:

- Start transaction
- Set transaction
- Test completion

- Savepoint
- Release savepoint
- Commit
- Rollback
- Connect
- Set connection
- Disconnect statement

7. SQL:1999's Impact on the Relational Data Model

As can be seen from the list of capabilities presented in Section 6, SQL:1999 is no longer a simple language for defining, accessing and managing tables consisting only of single valued columns of data. With respect to the basic data model capabilities, the SQL:1999 language more closely supports the independent logical file data model from the 1960s. It is therefore true to say that SQL:1999 is more of an implementation of the independent logical file data model (e.g., Adabas, Inquire, Datacom/DB, and Sybase) than of the 1970 relational data model.

SQL:1999 has, however, gone way beyond the capabilities of the independent logical file data model by incorporating facilities such as user-defined types, embedded programming language, and libraries of SQL:1999 defined routines for areas like full text management and spatial data. To say that these SQL:1999 extensions are mere “extended interpretations” of the relational data model is like saying that an intercontinental ballistic missile is merely an “extended interpretation” of a spear.

SQL:1999's impact on network and hierarchical data model DBMSs is significant. Network data model DBMSs have traditionally allowed complex data record structures with arrays, groups, repeating groups and nested repeating groups. A very unique characteristic of the SQL:1999 data model is that it now allows arrays. In addition, the elements of the array are able to be outward references to other data. Since the order of the elements in an SQL:1999 array is maintained by the SQL:1999 DBMS, then the array, with its outward references, is essentially a CODASYL set. This is a dramatic departure from the relational data model.

The only remaining and viable network DBMSs are IDMS by Computer Associates and Oracle DBMS (formerly the VAX DBMS). Both have had an SQL language interface for about 10 years. How Computer Associates plans to take advantage of the existing IDMS facilities with SQL:1999 is not known. A significant customer of Oracle’s DBMS (formerly Vax DBMS from DEC) is Intel who uses the Consilium manufacturing package to manage computer chip manufacturing. How the Oracle Corporation plans to take advantage of the existing VAX DBMS facilities with SQL:1999 is also not known

The only two hierarchical DBMSs, System 2000 and IBM's IMS will likely not be impacted at all. System 2000 is no longer being advanced by SAS, and IBM has a full implementation of DB/2 on many different operating systems.

SQL:1999's impact on independent logical file DBMSs, for example, Adabas, Focus, and Datacom/DB is significant. These DBMSs already support many of the SQL:1999 data model facilities. It would seem that these DBMSs could rapidly conform to the new SQL:1999 standard. If these vendors embrace the SQL:1999 model, then these DBMSs could claim conformance sooner than the existing set of relational DBMSs.

Simply stated, the SQL:1999 language defines a unique data model. It contains:

- The ability to model CODASYL sets,
- Many of the natural data clustering features of the hierarchical data model,
- Explicit many-to-many and inferential relationships like the independent logical file data model, and finally,
- The unique ability to directly model recursive relationships.

It therefore can only be said that the SQL:1999 data model is unique unto itself. Clearly, it is not the relational data model, CODASYL network, hierarchical, or independent logical file data models. Simply, SQL:1999 is a data model unto itself.

8. SQL:1999's Impact on Database Applications

For the past 20 years, database designers and implementors have struggled with highly normalized databases that perform poorly. The only solution is to denormalize by collapsing hierarchies of non-redundant tables into a single flat table with replicated data. While these highly redundant collapsed tables speed data reporting, it slows updating, and also becomes a significant risk for data integrity. That is because the data is highly disbursed and is duplicated across these report-tuned denormalized database structures that are commonly known as data warehouses. For all these reasons, most organizations only allow reporting from data warehouse databases.

As DBMS vendors implement SQL:1999, the database design process will transform itself from designing third normal table designs and then denormalizing these tables to enable cost effective reports to a set of database design activities similar to the ones that were commonly performed in database design efforts of the middle 1970s through the middle 1980s. There will have to be a greater knowledge of the application's processing to take advantage of the natural data structure hierarchies now possible within SQL:1999 tables.

While processing speeds will dramatically improve with SQL:1999 conforming DBMSs, the effort and processing time effort required to accomplish database redesigns and reorganizations will dramatically increase.

In short, we are returning to the past. That is, the data structures of the network and independent logical file DBMSs. While we will see increased performance for well designed and highly tuned databases, we will also see the return of significant designer and analyst time for database design and redesigns.

Keith Hare of JCC Consulting (www.jcc.com), a long time member of H2 and a user of Vax DBMS products put it best when he said, "With SQL:1999 you can get the best of both worlds and of course, you can get the worst of both worlds. It is up to the database practitioners to do the right thing."