RELATIONAL DATA BASE: HOW DO WE KNOW IF WE NEED ONE?

Orland Larson
Hewlett-Packard Company, Cupertino, California, USA

Summary

The field of relational technology is clearly misunderstood by a large number of people. One major obstacle to acceptance of the relational model is the unfamiliar terminology in which relational concepts are expressed. In addition, there are a number of misconceptions or "myths" that have grown up in the past few years concerning relational systems. The purpose of this paper is to define those terms, correct some of those misconceptions and to help you decide if your company can benefit from adding relational data base technology to your current capabilities.

This paper reports on the growing body of knowledge about relational technology. It begins by reviewing the challenges facing the MIS organization and the motivation for relational technology. It then briefly describes the history of relational technology and defines the basic terminology used in the relational approach. This will be followed by an examination of the productivity features of the relational approach and why it should be seen as a complement rather than a replacement for existing network databases such as the IMAGE data base management system. Typical application areas where the relational approach can be very effective will also be surveyed. Finally, a checklist will be reviewed that will help the audience determine if, indeed, they really can benefit from using a relational data base.

Introduction

The Challenges Facing MIS

The MIS manager is facing many challenges in today's modern information systems organization. The backlog of applications waiting to be developed is one of key challenges concerning MIS. In most medium to large installations the backlog of applications waiting to be developed is anywhere from two to five years. This estimate doesn't include the "invisible backlog," the needed applications which aren't even requested because of the current known backlog. Software costs are increasing because people costs are going up and because of the shortage of skilled EDP specialists. The data base administrator is typically using non-relational data bases where a great deal of time is spent predefining data relationships only to find that the users data requirements are changing dynamically. These changes in user requirements cause modifications to the data base structure and, in many cases, the associated application programs.

The application programmer is spending a significant amount of time developing applications using these non-relational data bases, which require traversing or navigating the data base. This results in excessive application development time. Because the users requirements change dynamically, it also means a great deal of time spent maintaining applications. The programmer is also frequently restricted by the data structures in the data base, adding to the complexity of accessing data.

End users or business professionals are frustrated by the limited access to information that they know exists somewhere in the data base. Their business environment is changing dynamically, and they feel MIS should keep up with these changes. They find the applications are inflexible, due to the pre-defined relationships designed into the data base. They also lack powerful inquiry facilities to aid in the decision-making process, which would allow them to ask anything about anything residing in that data base.

The Motivation for Relational

Dr. Codd, considered to be the originator of the relational model for data bases , noted when presented the 1981 ACM Turing Award, that the most important motivation for the research work resulting in the relational model was the objective of providing a sharp and clear boundary between the logical and physical aspects of data base management (including data base design, data retrieval, and data manipulation). This is called the data independence objective.

A second objective was to make the model structurally simple, so that all kinds of users and programmers could have a common understanding of the data, and could therefore communicate with one another about the database. This is called the communicability objective.

A third objective was to introduce high level language concepts to enable users to express operations on large chunks of information at a time. This entailed providing a foundation for set oriented processing (i.e., the ability to express in a single statement the processing of multiple sets of records at a time). This is called the set-processing objective.

Another primary motivation for development of the relational model has been to make data access more flexible. Because there are no pointers embedded with the data, the relational programmer does not have to be concerned about following pre-defined access paths or navigating the database, which force him to think and code at a needlessly low level of structural detail.

The Relational Data Model: A Brief History

In 1970, Dr. E.F. Codd published an article in the Communications of the ACM entitled "A Relational Model of Data for Large Shared Data Banks." This classic paper marks the "birth" of the relational model. Dr. Codd was the first to inject mathematical principles and rigor into the study of database management.

By the mid 70's, there were two database prototypes being developed. IBM was behind a project called "System R," and there was another relational database being developed at the University of California, Berkeley called INGRES. It was late 1979 before the first commercially available relational database arrived in the marketplace called ORACLE, from ORACLE Corp., which was an implementation based on System R. In 1981 Relational Technology Inc. introduced INGRES which was a different implementation based on the research done at Berkeley. Today there are several additional advanced relational products available, such as SQL/DS and DB2 from IBM and Rdb from Digital Equipment Corporation. There are additional products sometimes referred to as "born again" relational databases such as IDMS/R from Cullinet, ADR's DATACOM/DB, and Software AG's ADABAS, to name a few.

Relational Database Defined

The relational database model is the easiest one to understand - at least at the most basic level. In this model, data are represented as a table, with each horizontal row representing a record and each vertical column representing one of the attributes, or fields, of the record. Users find it natural to organize and manipulate data stored in tables, having long familiarity with tables dating from elementary school.

The Table, or two dimensional array, in a "true" relational data base is subject to some special constraints. First, no row can exactly duplicate any other row. (If it did, one of the rows would be unnecessary). Second, there must be an entry in at least one column or combination of columns that is unique for each row; the column heading for this column, or group of columns, is the "key" that identifies the table and serves as a marker for search operations. Third, there must be one and only one entry in each row-column cell.

A fourth requirement, that the rows be in no particular order, is both a strength and a weakness of the relational model. Adding a new item can be thought of as adding a row at the bottom of the table; hence there is no need to squeeze a new item in between preexisting items as in other database structures. However, to find a particular item, the entire table may have to be searched.

There are three kinds of tables in the relational model: base tables, views, and result tables. A base table is named, defined in detail, filled with data, and is more or less a permanent structure in the database.

A view can be seen as a "window" into one or more tables. It consists of a row and/or column subset of one or more base tables. Data is not stored in a view, so a view is often referred to as a logical or virtual table. Only the definition of a view is stored in the database, and that view definition is then invoked whenever the view is referenced in a command. Views are convenient for limiting the picture a user or program has of the data, thereby simplifying both data security and data access.

A result table contains the data that results from a retrieval request. It has no name and generally has a brief existence. This kind of table is not stored in the database, but can be directed to an output device.

The Relational Language

The defacto industry standard language for relational data bases is SQL. SQL stands for Structured Query Language. This name is deceiving in that it only describes one facet of SQL's capabilities. In addition to the inquiry or data retrieval operations, SQL also includes all the commands needed for data manipulation. The user only needs to learn four commands to handle all data retrieval and manipulation of a relational database. These four commands are: SELECT, UPDATE, DELETE and INSERT.

The relational model uses three primary operations to retrieve records from one or more tables: select, project and join. These operations are based on the mathematical theories that underlie relational technology, and they all use the same command, SELECT. The select operation retrieves a subset of rows from a table that meet certain criteria. The project retrieves specific columns from a table. The join operation combines data from two or more tables by matching values in one table against values in the other tables. For all rows that contain matching values, a result row is created by combining the columns from the tables eliminating redundant columns.

The basic form of the SELECT command is:

```
SELECT      some data (field names)
FROM        some place (table names)
WHERE       certain conditions (if any) are to be met
```

In some instances WHERE may not be neccessary. Around this SELECT..FROM..WHERE structure, the user can place other SQL commands in order to express the many powerful operations of the language.

In all uses of SQL, the user does not have to be concerned with how the system should get the data. Rather, the user tells the system what he wants. This means that the user only needs to know the meaning of the data, not its physical representation, and this feature can relieve the user from many of the complexities of data access.

The data manipulation operations include UPDATE, DELETE and INSERT. The UPDATE command changes data values in all rows that meet the WHERE qualification. The DELETE command deletes all rows that meet the WHERE qualification and the INSERT command adds new rows to a table.

When retrieving data in application programs it is important to remember that SQL retrieves sets of data rather than individual records and consequently requires different programming techniques. There are two options for presenting selected data to programs. If an array is established in the program, a BULK SELECT can retrieve the entire set of qualifying rows, and store them in the array for programmatic processing. Alternatively, it is possible to activate a cursor that will present rows to programs one at a time.

SQL has a set of built-in, aggregate functions. The functions available are count, sum, average, minimum, and maximum. They operate on a collection of values and produce a single value.

In addition to commands for data retrieval and modification, SQL also includes commands for defining all database objects. The data definition commands are CREATE, ALTER and DROP. The CREATE command is used to create base tables and views. The ALTER provides for the expansion of existing tables and the DROP deletes a view. One of the most powerful features of SQL is its dynamic definition capability. This function allows the user to add columns, tables and views to the database without unloading and reloading existing data or changing any current programs. More importantly, these changes can be made while the databases are in use.

Productivity Features of Using Relational Technology

Relational technology is one very important tool that can contribute to making data processing professionals more productive. The programmer can benefit from a facility called interactive program development, which allows the development and debugging of SQL commands and then permits the moving of those same commands into the application programs. It is convenient and easy to set up test databases interactively and then to confirm the effect of a program on the database. All of these characteristics make SQL a powerful prototyping tool. The on-line facilities of SQL can be used to create prototype tables loaded with sample or production data. On-line queries can easily be written to demonstrat application usage. End users can see the proposed scheme in operation prior to formal application development. In this prototype approach, people-time and computer-time are saved while design flaws are easily corrected early in development.

The data base administrator profits from the productivity features already described for programmers. The database administrator has a great deal of freedom in structuring the database, since it is unneccessary to predict all future access paths at design time. Instead, the DBA can concentrate on specific data requirements of the user. Nonrelational models, on the other hand, require all relationships be pre-defined, which adds to the complexity of the application and lengthens development time.

Additional productivity features for the database administrator include the capability to modify tables without affecting existing programs and the capability to dynamically allocate additional space while the database is still in use. SQL goes far beyond many database management systems in the degree of protection that it provides for data. Views make it possible to narrow access privileges down to a single field. Users can even be limited to summary data. Protection can be specified for database, system catalog, tables, views, columns, rows and fields. It is also possible to restrict access to a subset of commands. These access privileges can be changed dynamically, as the need arises.

In many installations, the key to overall productivity is the ability of DP too offload the appropriate portions of the development and maintenance to the end user. The flexible design approach of relational databases allows an application to be designed with the end user's requirements in mind. This could enable the DP professional to implement an application up to the point where the end user could create and execute his own queries, thereby expanding the application on his own and reducing his dependence on the data processing department. Through SQL, the end user is provided with extremely flexible access and simple but powerful commands.

Relational and Nonrelational: Complementary Technologies

Within a data processing department already using a well-established nonrelational DBMS, what role can relational technology be expected to play? We know that DP will not automatically drop everything and go to relational. Rather, relational technology should be seen as a complement rather than a replacement for nonrelational database systems. Both approaches offer a host of benefits, and most applications can be implemented with either of the two.

The relational approach is preferred when the application has a large number of data relationships or when the data relationships are unknown or changing dynamically. The relational approach provides the needed flexibility to establish relationships at the time of inquiry, not when the database is designed. If the application has unknown of incomplete data specifications, which is usually the case in a prototyping environment, then a relational system may be preferable. If the application requires a quick turnaround, the quick design and implementation capabilities of a relational database can be important. The ability to handle ad hoc requests is a definite strength of the relational model as is the ability to extract data for use in a modeling, forecasting, or analytical framework.

The nonrelational approach is preferred for high-volume on-line transaction processing applications where performance is the most critical requirement.

Choosing the Right Technology

The choice of the "correct" database management system must be based on the environment in which the database will be used and on the needs of the particular application. The key feature of relational technology is that it allows for maximum flexibility, and will probably be the choice for many new applications. On the other hand, nonrelational systems may continue to be preferrable for very stable or structured applications in which data manipulation requirements are highly predictable, and high transaction throughput is important.

The optimum approach for many MIS departments will be to use the relational system concurrently with the existing nonrelational system, matching the appropriate technology to the application. The only problem with such an approach is that the data for an application developed in one technology may sometimes be needed by applications developed in the other technology. Data may be "locked out" from an application that needs it, or users might be tempted to duplicate the data, maintaining both copies. The most desirable solution would obviously be to provide both relational and nonrelational access to a single database.

Relational Applications

There are many application areas - particularly those involving user analysis, reporting, and planning - where the very nature of the application is constantly changing. Some typical application areas are:

* Financial
  - Budget analysis
  - Profit and Loss
  - Risk assessment

* Inventory
  - Vendor performance
  - Buyer performance

* Marketing and sales
  - Tracking and analysis
  - Forecasting

* Personnel
  - Compliance
  - Skills and job tracking

* Project management
  - Checkpoint/milestone progress
  - Development and test status

* EDP auditing
  - Data verification
  - Installation configuration

* Government/education/health
  - Crime and traffic analysis
  - Admissions/recruiting/research
  - Medical data analysis

These applications typify instances where it is of primary importance to establish interrelationships within the database and to define new tables.

Checklist for Deciding Whether or Not You Need A Relational Database

Note: If you answer yes to any of the following questions, you should seriously consider taking advantage of relational technology.

1. Does my company have an excessive backlog of applications to be developed, including an invisible backlog?

2. Are we spending too much money developing applications due to the complexities of using non relational systems?

3. Do our users' requirements for information change dynamically?

4. Are we spending too much time maintaining applications caused by changing data requirements or relationships?

5. Do our users feel restricted by a non-relational database?

6. Are programmers spending an excessive amount of time writing code to navigate through nonrelational databases?

7. Is the nature of our applications such that it is constantly changing?

8. Would your users find it natural to organize and manipulate data in tables?

9. Do your users currently use LOTUS 1-2-3 or spreadsheets?

10. Is your company moving towards a true distributed database environment?

Bibliography

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM, 13 6,(June 1970),pp. 377-387.

Codd, E.F.,"Relational Database: A Practical Foundation for Productivity," CACM, 25 2,(February 1982,pp. 109-117.

Date, C.J., An Introduction to Database Systems. Addison-Wesley, 1977.

_____,Relational Technology: A Productivity Solution, Hewlett-Packard Co., Computer Systems Division,Cupertino,Ca.,5954-6676,January 1986.

Biography

Orland Larson
is currently Information Resource Management Specialist for Hewlett-Packard. As the database and application development specialist for the Information Systems Tactical Marketing Center he develops and presents seminars worldwide on database management, information systems prototyping and productivity tools for information resource management. He is a regular speaker at Hewlett-Packard's Productivity Shows and Users Group Meetings and also participates in various National Data Base and 4th Generation Language symposiums. Previously he was the Product Manager for IMAGE/3000, Hewlett-Packard's award winning database management system.

Before joining HP he worked as a Senior Analyst in the MIS Department of a large California-based insurance company and prior to that as a Programmer/Analyst for various software companies. Mr. Larson has been with Hewlett-Packard since 1972.