VMARK
SOFTWARE

WHITE PAPER
uniVerse™ - A Relational Database and More

Len Greenwood
Principal Technical Consultant
VMARK Software, Inc.

Introduction

With the advent of uniVerse Release 7, a number of the arguments that data purists had with the notion of uniVerse as a true relational database are no longer valid. This paper describes why uniVerse in its current form bears comparison with today's 'traditional' relational databases, and where it has the capacity to provide a superior alternative - the best of relational - while offering its own real-world advantages.

Why are Relational Databases so important?

The invention of the relational model has been described as the single most important development in the entire history of databases. It provided a sound theoretical data model on which databases could be designed. Previous data models, such as networked or hierarchical (tree-based) models, were generally defined 'after the event' by abstracting them from specific storage architectures.

The relational model also set out goals for database manager implementations, such as providing data independence from programs; end-user and programmer interface consistency; and security and integrity rules. The ultimate aim of the model was to improve productivity for all those involved in data creation, manipulation, maintenance and administration - that is, database designers, end-users, programmers and DBAs.

Tables, Tables Everywhere...

A relational database is perceived by its users as a collection of *tables* (and nothing but tables). The database system provides operators that can be used to generate new tables from old. For example, one operator will extract a subset of the rows of a table, and another will extract a subset of the *columns*. These subsets also can be regarded as tables in their own right, and further operated on to produce new tables. And more complex operators can be used to form relationships between the various tables - these relationships, of course, being viewed as yet more tables.

Tables (also called *relations*) are essentially two-dimensional objects whose rows are uniquely identified by a primary key. Relationships between tables are achieved by using values in the columns of one table as keys to another. Unlike other types of databases, these relationships are not stored as physical pointers anywhere - they are built up, as requested, by the user. One of the major advances in relational database research was the development of the theory of normalization, which helps to produce databases with no redundancy or logical anomalies. This ensures that any requested relationship can be derived at a later date without losing or duplicating information.

Easy to Use

Simply organizing data into tables is not sufficient to qualify a database as relational. The database is also characterized by having a system-wide interface language and various integrity features. The language ensures that there is a single, consistent way of performing all operations on the database.

In practice, the language chosen is usually SQL (*Structured Query Language*), which, in its more recent versions, has the advantage of a large body of standardization behind it. SQL is a complete command-line interface for creating, modifying and manipulating the database itself, with built-in features for handling security and integrity constraints. SQL is widely recognized, and therefore portable; it is well-defined; and it is a complete, in the sense that ideally no other language need be learned to perform database tasks.

However, the presence of SQL does not mean that this is the only language that can be used to address the database (after all - most people will admit that SQL is not the most user-friendly language ever invented). Other interfaces can be supplied - oriented toward different user needs perhaps - so long as they do not provide a means of subverting the underlying relational database model.

Improving Productivity

SQL is basically a *non-procedural* language. This is, it presents the database to the user as a high-level abstraction rather than requiring him or her to know physical file layouts or storage details. In fact, it does not allow the user to navigate the database at all, in the sense of earlier, non-relational databases. This onus is on the database manager, rather than the programmer or end-user, to decide the best means of fulfilling a request. Over time, a large body of knowledge has been built up concerning algorithms to optimize SQL queries on a relational database. In general, the database manager's plans for retrieving information will be as efficient as that which a skilled programmer would have had to produce by hand for a non-relational database.

This level of abstraction, combined with the flexibility and sound design principles of the underlying data model, means that database requests that might previously have taken days of skilled programmer time can now be handled simply. In the worst case, such requests might even have been impossible to satisfy without restructuring the database.

uniVerse as a Relational Database

Some of uniVerse's traditional strengths has always been its ease-of-use and immense flexibility. It aims to satisfy two of the primary criticisms directed at non-relational databases; namely, that they require a profound understanding of the data model being used, and that once in use, a database is hard to change. Recent developments to uniVerse have added those aspects of the relational security and data integrity model that were previously missing. They also supply, in SQL, access to the standard data manipulation language associated with relational databases.

The uniVerse Data Model

The uniVerse data model is an extended form of the relational data model, based originally on that pioneered by the PICK system. In this model, all data is held as tables, just as in a conventional relational database. In common with most other relational databases, tables are described by dictionaries, which hold logical descriptions of what the table's columns mean. A powerful feature, not found in most other relational databases, is the ability to use the same dictionary to describe several tables, or different dictionaries to describe the same table, as circumstances warrant.

In addition, uniVerse does not expect the data in a field to be of a defined length or type, or the fields in a record to be fixed in number. This means that all data and tables are very flexible as to size, and easily redefined. The physical data manager is optimized for dynamically-changing databases, and can, therefore, easily accommodate the rapidly-fluctuating requirements of commercial data processing, while maintaining efficient data access paths.

One important aspect of the traditional relational data model is that the data items that are stored in the row/column intersections of a table must be *atomic* values. That is, there is no further information to be derived from looking more closely at the value. This rule was initially made as a simplifying assumption when the relational algebra (the mathematics that underpin the relational data model) was first developed.

More recent work has determined that changing this assumption leads to some useful extensions to the data model. If one considers that a data value can itself be made up of *subvalues*, the notion of *repeating fields* arises quite naturally out of the model. A simple way to look at these is to consider a set of repeating fields within a table as an embedded table in its own right. Provide the embedded table satisfies the criteria for tables in general (for example, having a unique key), the operators of the relational algebra extend naturally.

The uniVerse data model supports *associated multi-valued fields*. Extensions to the SQL syntax allow these to be accessed as an extension to the relational model: and they can, of course, be accessed via the standard uniVerse tools, such as Retrieve, Revise, and BASIC. Repeating fields are a generally useful feature in the world of commercial data processing, and we will return to them later.

Where SQL Fits In

The implementation of SQL in uniVerse Release 7 adheres to existing ANSI standards for interactive SQL, with integrity enhancements (*ANSI X3.135-1989*). It also looks forward to enhancements proposed both by that group and others (*X/Open, SAG*). As permitted by these standards, uniVerse SQL has extensions for handling uniVerse-style dictionaries, BASIC code fragments in queries, and formatting and data conversion capabilities. This means that the database can be concurrently manipulated by uniVerse's own Retrieve statements or by standard-compliant SQL statements.

To support SQL, the underlying uniVerse database engine has been changed to support a number of new features including:

- Security - which users can access which tables and fields, and what operations they can perform on them.
- Entity integrity - What data values are permitted in participation fields.
- Referential integrity - ensuring that data at both ends of a relationship exists and is valid at all times.
- Null values - the ability to define a distinct data value with the meaning "unknown" as opposed to zero, or empty, and have all logical and arithmetic operators and functions deal correctly with such a value.

- Transactions - bounding a number of logically-related database updates so that they are all either applied or not applied as a group.

- Queries from multiple tables - These are now possible in a single SQL command where previously they would have required several Retrieve commands or writing a specific BASIC program.

- Subqueries - where the output from one query is automatically fed as parameters into a higher-level query.

So although SQL is not absolutely necessary in a relational database, supporting it leads naturally to a number of features that are essential if a database is truly to be considered relational.

What's Wrong with Relational Databases?

One of the limitations of the normalization approach expected by conventional relational databases is that it makes no allowance for repeating fields. Indeed, it absolutely cannot permit them due to the nature of its underlying data model. However, it is generally recognized that such fields occur very frequently in commercial applications where lists of items are often the natural way of describing things.

Many data design methodologies actually allow repeating fields to be defined and then removed as part of the normalization process. This part of the process converts tables to *First Normal Form*, or *1NF* for short. However, this removal of repeating fields in not necessarily good for the application. If the most usual form of access to the fields requires access to all its values, then a 1NF database will need to perform a *Join* operation each time to retrieve the appropriate values from another table where they have been stored. Holding the values physically in the fields can obviously lead to more efficient data access in these circumstances.

Tables that hold repeating fields are said to be in *Non-First Normal Form*, or *NF2* for short. As mentioned previously, provided that the fields involved obey certain rules that allow them to be treated as tables embedded within tables, NF2 data forms do not violate the tenets of relational algebra. Further, no data is hidden, since the extended operators that work on NF2 tables allow the embedded tables to be extracted and the data viewed as if it came from 1NF tables in the first place. In many cases, the latter form will be the exception rather than the rule. It will be more efficient to access the repeating fields concurrently with the data most of the time, knowing that it can be extracted and viewed as a separate table on the infrequent occasions when that may be required.

Communicating between Databases - Clients and Servers

SQL has become the standard mechanism for database interoperation. Once a database supports SQL fully, it can communicate with other database whose storage mechanisms may differ but whose SQL interfaces hide that. Subsequent releases of uniVerse will enable the sharing of data with non-uniVerse applications. The uniVerse server will permit a uniVerse client to request data from other SQL servers (such as ORACLE, Sybase or Informix) and will allow non-uniVerse clients to make SQL requests for data from a uniVerse database.