A Relational Database for Inventory and Asset Management

John Flatness

Thomas Jefferson High School for Science and Technology

Computer Systems Laboratory

2005-2006

Mentor:

Gary Carr

ENSCO, Inc.

Applied Technology and Engineering Division

Abstract

Background: ENSCO, Inc.'s Applied Technology and Engineering (ATE) Division is a large group with complicated inventory management needs. Each employee has, in general, one laptop computer and one desktop computer, and several have multiples of either one or both of these, and additionally many have personal printers and scanners. To further confuse matters, each employee has a different set of software packages installed, many with single-use licenses restricting their use to one computer. This set of circumstances makes it vital to know which pieces of hardware each employee has, and which software packages are associated with those pieces of hardware. Existing inventory management systems in place are sparsely deployed, and clumsy to update.

Description: Currently, the ATE Division's inventory and asset management information is decentralized, with data stored in varying locations and in differing formats. Such data that is stored electronically tends to be out of date and of little use. In an attempt to unify the system for the storage, retrieval, and maintenance of this information, the current information will be migrated into a single relational database. Instead of one large all-encompassing table of values, this database will contain separate tables for data about users, equipment, and software, thus allowing for each set of data to be changed without upsetting the integrity of the other two.

In order to ensure easy access and convenient maintenance, the chosen database must have a front-end suitable for both retrieval and presentation of data as well as a system for editing existing values and adding new records. For this reason, the Microsoft Access relational database management system will be used, since it incorporates Microsoft's Jet database engine with an editing front-end and a system for various presentations of the data. This new system will allow a centralized electronic location for inventory management, as well as a simple process for producing paper copies of the relevant information.

1. Introduction

Inventory and asset management can frequently be a difficult task for any organization. In a typical situation, the organization's assets are widely distributed among all of its employees, making the proper maintenance of any lists or systems nearly impossible. To make matters worse, employees are often added, leave, change locations, and the same can be said for the equipment itself. This means that it is vitally important for there to be one central location where all asset management informations can be stored, and easily updated. The nature of this requirement quickly rules out any non-electronic or hard-copy forms of information storage, as the number of required updates to keep such a listing current would quickly become unworkable.

Even among electronic storage formats, there are a myriad of choices. A simple spreadsheet could easily keep listings of assets, along with other pertinent information such as serial numbers, and which user currently has the asset in question. However, it is simple to note that this system also suffers from difficulties in updating. Simply changing one user, who may be associated with many assets, would require editing a field for every asset that user was associated with. Therefore, a more desirable format is a relational database. This format allows the data for users, hardware, and software to be stored in separate tables (or, in the relational model, *relations*) so that they may be maintained independently (Codd, 1970).

Since a relational database is the chosen format for data storage (the back-end), a user interface (front-end) to the data must be chosen or implemented. Traditionally, the front-end for a relational database is a software package called a relational database management system, or RDBMS. Such systems allow for direct access to the underlying data, generally in the format of tables and relationships that are editable by the user, and create links between the different sets of data that are stored within. Once these links have been properly established, the user can cross-reference the different sets, both for purposes of data validation and for data retrieval.

This paper focuses on the particular implementation of the relational database model for one organization, ENSCO Inc.'s Applied Technology and Engineering (ATE) division. While some of the following information will therefore be specific to this implementation, the process in general remains valid for almost any similar asset management requirements.

2. Background

The relational database model is characterized by a separation of data into several different relations. A relation is the mathematical term for what is often represented as a table of data. Thus, data will often be described in terms of "rows" and "columns". Rows are specific records, a user or an asset to use this implementation as a model. Columns are groupings of data contained within the rows. Continuing the users example, one column would be the user's name, one his room number, one his ID number. The model thrives on this separation, combined with the process of cross-referencing. For the model to achieve its true potential, the separate tables must be linked in some way, so that data from one table can be used to retrieve associated data from another.

In order for these *relationships* to be made, each row of data must be uniquely identified in some way, so that there is a key from which data can be retrieved. This is required because the ordering of the rows is unimportant, and can be changed at will (Codd, 1970). Therefore, one of the columns must be reserved for this unique identifier in each column. The choices for what data to use for these unique identifiers, as well as which other columns should be included in the tables, and how the data itself can be effectively divided into separate tables, define the nature of the resulting database, and can have a huge impact on its effectiveness.

3. Development

3. 1. Design Criteria

The inventory and asset management database was required to fulfill several criteria in order to be suitable for use. Chief among these requirements were the following:

- The data must be stored in one central, documented, location. One of the most important goals of this project is to ensure that the inventory data would be collected and condensed into one location.
- The process of updating the database must be simple and quick. The largest barrier to the long-term success of the asset management system is the fact that the data must be frequently updated to be relevant. As a consequence, updates must be a simple process, so that the time invested in maintenance is as trivial as possible.
- The database must be extensible. A database setup with rigid rules about what types and numbers of assets and data can be stored will not be able to adapt to changes in technology and the needs of a corporation, and is as such unacceptable. As much as possible, the system should allow different types of assets to store different types of data, as is required.

All of these requirements had to be taken into account when making decisions and selections regarding both the form of the database and the specific software used to manage the data and provide it to users and administrators.

3.2. Materials

As was previously mentioned, after the decision to use a relational database was made, two extremely important choices had to be made. Both a back-end, the relational database itself, and a front-end or user interface to the database had to be selected. In general, the simplest way to make these choices is to combine them into one choice of a relational database management system, or RDBMS. A RDBMS combines the back-end and the front-end into one program, so

that to the casual user, the back-end is essentially invisible, but an administrator can still have low-level access to the internals of the underlying relational database.

Among the most popular RDBMS software packages are MySQL, PostgreSQL, and two packages from Microsoft, SQL Server and Access. The first three in this list have an important feature in common. While all of the listed packages do contain a user interface, as is required by the definition of a RDBMS, the first three only contain a command-line or text-only interface by default. While the command-line interface can be very effective in the hands of a skilled user, the purpose of this project was to create a system which could be used by relatively untrained users. An important step towards achieving this goal was the use of a graphical user interface, or GUI. Of the packages on this list, Microsoft Access is the only one which contains a GUI, and in addition, the software had nearly 100% deployment already throughout the ATE Division as part of the Microsoft Office suite of programs. For these reasons, Microsoft Access, and it's underlying database system, the Microsoft Jet Engine, were selected as the basis of the project.

3.3. Procedures

To take full advantage of the relational database model, several tables or relations must be established, with the data in these tables linked in some way. In Access, the linking between these tables is done through *relationships*. Each table can derive some of the information contained within it from any other table in the database, or from another database entirely. The caveat to successfully implementing relationships is that each table must have one column whose data can be used to uniquely identify the rest of the data in the row. This column of data is called the *primary key*.

In this implementation, there were two tables established within the database. The first, called Users, contained the names and room numbers of all of the ATE division employees, along with an automatically assigned unique ID number, which served as the primary key. The

second table, named Assets, contained information about the computer and computer-related assets in the division. Namely, these included type of asset, manufacturer, model name, serial number, software (in the case that the asset was a type of computer), and, most importantly, a reference to the user who "owned" the asset in question. Also included was, similar to the implementation of the Users table, an automatically assigned ID that would serve as the primary key.

To facilitate the linkage of the Users and Assets tables, a relationship was created between the two, with the Users table being the "master" table and the Assets table being the "slave." This setup means that each asset, when added to the table, must have a user associated with it. Due to the relationship between the tables, Access checks the associated user to ensure that said user is contained within the Users table, thus enforcing referential integrity.

4. Results and Discussion

The linkage, or relationship, between the Users and the Assets tables provides several advantages. First, as was mentioned above, the relationship allows for the enforcement of referential integrity, ensuring that users associated with assets actually exist. In addition, the relationship allows lookups to be performed in the other direction, from the Users table.

Since each member of the Assets table must have an associated member from the Users table, lookups on specific users can be performed to retrieve all of the assets owned by that particular user. Access's implementation of this feature of the relational model is shown through nested tables, or sub-tables. Each user, assuming that they have associated assets, will have a sub-table which shows a miniature version of the Assets table which only contains those assets associated with the user in question. This means that users of the database can quickly use the Users table to obtain all the information stored about a user, including their assets. At the same time, they can access the more "raw" form of the information, stored in the full Assets table.

Thus, the database is well suited to several different styles and needs of information retrieval. The same holds for maintenance of the database. Just as users can access the data in several different ways, so can the administrators modify it in ways that parallel those previously mentioned. A deleted user can have all of his assets erased from the table (an unlikely situation), or the assets can be reassigned before the user's deletion.

5. Conclusion and Recommendations

The established relational database is effective for allowing independent changes to the set of users and the set of hardware and software, and is a centralized version of several other sets of data. However, this system, while in many ways an improvement over the previous systems, could itself be improved, especially in the area of automation. Currently, the database maintains a Users table which lists all of the users in the division, which in practical terms means that every employee is in the Users table. This means that the asset database could be linked with the pre-existing database for personnel and payroll, which would mean that the list of users would automatically be updated upon personnel changes within the division, further increasing the ease of database maintenance.

Also, the primary keys chosen, while sufficient to satisfy the requirements for using the relational database model, were less than optimal. Both tables utilized an automatically assigned primary key which had no significance outside the database. A better solution would have been to use existing unique identifiers for primary keys, allowing the database to easily be related to other information sources. For the Users table, a suitable key would have been the Employee ID, a number which is assigned to each employee, and is by nature of its purpose a unique identifier. Unfortunately, I did not have access to this particular piece of data. Similarly, a useful identifier for the Assets table would have been a number assigned to the asset when it was acquired. It appeared that no such global identifier existed. Interestingly, the process could be reversed, and

the automatically assigned database ID could be used as the global identifier, thus achieving the same results.

In addition, while the Microsoft Access relational database management system is a very convenient combination of several database tools, integrating the database itself with the user interfaces for data editing and display, there are several ways of improving ease of use for all parties involved. Most notably, migrating from a local application-based system, in this case Access, to a web-based system hosted on the company's intranet would allow users and administrators to use the database from almost any connected location, without any prior installation. Access is ill-suited to this type of deployment, and without the front-end benefits of Access, the use of the Jet database engine becomes less desired. The preferred form of this intranet-based system would involve migrating the database to some type of SQL (Structured Query Language) server for the back-end (possibly Microsoft SQL Server, MySQL, or PostgreSQL), and a front-end written in a combination of HTML and a scripting language such as PHP or Ruby, both of which have tools for interacting with SQL-based databases. A web interface is generally even more intuitive to the average user than the nested-table system employed by the Access GUI.

6. References

Codd, E. F. (1970, June). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM, 13*(6), 377-387.

Date, C. J. (2005). *Database in depth : relational theory for practitioners*. Cambridge: O'Reilly.