The relational database structure in Release 6.06 of the SAS® System has given the programmer new options in database design. Indexing allows quick access to data even when databases are very large. SQL® support provides the ability to create and store views on one or more data sets which can then be referenced as real data sets. This makes data access much simpler for users of complex systems. The fact that more data can be stored on a given amount of disk space using data compression and expanded on-line database utilities available under an enhanced SAS Display Manager makes managing large database applications much easier. These options and others make database design at once easier and more complex.

It is important to note that Release 6.06 does not represent a change in direction. High volume, on-line systems are still best developed as a single integrated system. However, applications that are updated through batch processing and used primarily for data analysis, the normalization exercise is instructive in that the desire for array processing, database simplicity, or large efficient bulk data retrieval, the transaction-counts-per-minute are required. But as more organizations recognize the value of formal information systems that have their own set of special requirements, Release 6.06 of the SAS System should find growing acceptance as a database management system particularly suited to these applications.

Design approaches for transaction versus information systems are as different as the features of the DBMSs needed to support them. Normalization is one design technique whereby the relationships between the data elements identified in the business model are analyzed and a data model is developed, where the data elements or variables are sorted into different tables or data sets, based on their relationship. This process plays an important part in ensuring data integrity and is becoming an increasingly common approach to database design for on-line transaction-based systems. However, applications that are updated through batch processing and used primarily for data analysis do not present the same data integrity issues that transaction systems carry with them. The advantages of high levels of normalization can sometimes be outweighed by other considerations such as efficient bulk data retrieval, the desire for array processing, database simplicity, or large batch update processing. But regardless of the nature of the application, the normalization exercise is instructive in that several design options can usually be revealed in the process. This paper balances the advantages and disadvantages of these design options when creating different types of applications.

Working Out a Database Design

For a lot of SAS System users, database design has been a seat-of-the-pants affair in the past. Most designs have been for data analysis projects where the data were discarded afterward, or small local systems where performance was not an issue and any one of several approaches could have been used. But even with these casual approaches it has been common for programmers to design partially normalized structures without being aware of the normalization process itself. By working through examples of the first three normal forms plus a de-normalized structure it is hoped that these programmers will either gain some support for their intuition or perhaps give them a few good ideas for alternative approaches.

Whether you are designing a system with a database for personal use or for an office party, it will usually save time in the long run to start with a business-oriented plan, progress to a design or model, and then write the code. The plan should include a written statement of purpose with a broad scenario on how the system is to be used. A second part should identify the specific environment and processes and the items, with their attributes, necessary to support those processes. At this point a data model of the system can be developed. With most smaller decision-support systems this model can be used directly to build the database definitions. With larger systems a logical model that reflects the flow of business and does not account for technical constraints and a physical model based on the actual technical implementation of the database separate the issues of how the system will meet the needs of the business and the technical constraints involved. Obviously there can be irreconcilable differences between the two but, at least, you have isolated the problem and can make adjustments in the scope of the business plan if necessary. Models are a lot cheaper to change than actual systems and, as will be shown below, a lot of different models can be built with the same data.

The example used in this paper is from a regional airline reservations office sales tracking system. Hand-written reports are produced by a team of agents that are temporarily assigned to the monthly project. They combine data from a system sales transactions report which carries only individual sales data identified by an agent ID, and a local scheduling system which carries personnel and scheduling information. This process takes up to 80 hours a month. The system report is available electronically once a month. The scheduling system is a simple on-line system based on a single file which includes multiple schedules. This system was designed by a person that has since been promoted and the clerks currently responsible for its maintenance have had some minor problems with it. Modifying this scheduling system for improved maintenance is a part of the sample project.

The plan for the sales tracking system includes a monthly report for each agent and the agent's supervisor. Sales per 100 calls per individual are measured against those agents with the same supervisor and schedule - an adjustment that is necessary since agents sometimes must deal with numerous flight information calls during foggy weather. This report is to show the past month's performance plus a history by month for the calendar year. This history is also used to generate reports for ad hoc analysis. A typical ad hoc activity might involve a cost/benefit analysis for creating a special desk or automated touch-tone flight information system which would free highly trained agents for assisting customers in their flight plans.

The elements of the sales tracking system are

<table>
<thead>
<tr>
<th>AGENT NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT ID</td>
</tr>
<tr>
<td>SUPERVISOR</td>
</tr>
<tr>
<td>TEAM</td>
</tr>
<tr>
<td>MONTH</td>
</tr>
<tr>
<td>SHIFT</td>
</tr>
<tr>
<td>SALES/CALLS</td>
</tr>
<tr>
<td>AVERAGE SALES/CALL/SHIFT</td>
</tr>
<tr>
<td>SALES/CALLS</td>
</tr>
</tbody>
</table>

Obviously some of these elements are derived from primary data. One of the problems in designing an historical database is how to store these elements.
The sales tracking table above includes enough primary data to satisfy ad hoc queries and one derived field that will also simplify ad hoc queries as well as basic reporting requirements. It also addresses the problem of an agent being dropped from the database and changing past average sales / 100 calls / shift numbers. Of course this could be addressed by not dropping agents in the first place.

The scheduling table reflects the current system. It reveals that maintenance issues might arise from the database design and not the supporting programs. If you suspect that the problem might be found with repeating patterns, you would have a good start.

**First Normal Form**

**Transforming Repeating Groups**

Repeating groups have often been intentionally designed into SAS data sets because of the array handling capabilities found in base SAS software. When data are presented in such a form such as might be found in a survey data file, array processing is a great tool. But in an on-line system such as the scheduling system SAS procedures could handle most of the data analysis requirements and usually with much less effort. The system would be much more flexible if repeating groups were eliminated. If NEXT MO shift data are desired or questions arise from a file restore as to what month's data is really in the file, the repeating group structure starts to show its limits. An optional design might break up the schedule table into an agent table and a schedule table along these lines:

**AGENT -**

AGENT ID
AGENT NAME
PHONE
MAIL STOP
SUPERVISOR

**SCHED -**

AGENT ID
MONTH
SHIFT

Second Normal Form

**Data Dependence on the Primary Key**

A note on primary and foreign keys:

Primary and foreign keys are the maps to a relational database. A primary key is a column (variable) or group of columns that have unique values for each row (observation). Foreign keys are columns that contain identical values to those found in the primary keys of other tables.

The second normal form removes columns from tables where the values found in those columns depend on only part of the primary key. The primary key for the sales tracking table includes the columns AGENT ID and MONTH. AGENT NAME is also carried in the sales tracking table but depends only on AGENT ID. In addition the relationship between AGENT ID and AGENT NAME is also carried in the agent table. Usually, data redundancy leads to problems with data integrity where a column will be updated in one place but its duplicate will not. You could remove AGENT NAME from the sales tracking table, but since that table will be updated once a month from the agent table via a batch job you may choose not to do so. If the link between these two columns changed over the course of the year, then the need for an AGENT ID, AGENT NAME, MONTH link in the sales tracking table or some other table would be required.

**Third Normal Form**

**Data Dependence Only on Primary Key**

The SUPERVISOR column is carried in several tables but as noted above a supervisor is assigned to a given shift each month. To implement a third normal form database this relationship - SHIFT, MONTH, SUPERVISOR - would be separated out into another table. In the case of a large transaction system you may see how this would ease maintenance issues, but again because the sales tracking table is updated on a batch basis and is intended to represent discrete points in time you may want to by-pass this form.

You may ask how, with such proliferation of tables, does one go about easily traversing a normalized database with any semblance of simplicity. This is where views come in. Views, or predetermined joins of a few or many tables, can make a normalized database ‘whole’ again. Depending on the task - editing, transaction access, or ad hoc query - one can choose the appropriate view. It is recommended that SAS programmers who use the normalization process study the SQL procedure.

**Denormalizing for Special Cases**

One of the assumptions made in the example above was that the scheduling table would be modified. This allowed the use of at least the first normal form for the sales tracking table. In times when it might be beneficial, that is for performance or single table/single row access to data for a given agent, the sales tracking table could be designed in a denormalized form such as:

**SALES TRACKING -**

AGENT ID
AGENT NAME
SUPERVISOR
MONTH01 SHIFT
MONTH01 SALES
MONTH01 CALLS
MONTH01 AVSALES/CALLS/SFT
MONTH02 SHIFT
MONTH02 SALES
MONTH02 CALLS
MONTH02 AVSALES/CALLS/SFT
and so on

Remember batch updates make the maintenance of such a table less of a problem than it might seem.
Conclusions

The types of databases that are usually built by SAS users are significantly different in purpose from the transaction systems that have, in the past, been the target of the majority of relational database design papers. Release 6.06 of the SAS System will allow us to use elements of this body of work but, we should be careful to choose only those which will benefit our efforts and discard those which do not.

For further study, "Interfacing Normalized Database Structures with SAS(R) Software", by James R. Johnson and Roger D. Cornejo of Glaxo, Inc., in SUGI 15 Proceedings is highly recommended.

SAS is a registered trademark of SAS Institute Inc., Cary, NC, U.S.A.

SQL is a trademark of International Business Machines Corporation