Creating and Managing a SQL* Database with the SAS® System
Mark Stranieri, SAS Institute Inc., Cary, NC

ABSTRACT
A database consisting of SQL procedure tables (SAS® data sets), PROC SQL views, and SAS indexes under Release 6.06 of the SAS System has been established. The process of creating and managing this SQL* database is discussed. Creating tables and views is best accomplished by using a combination of PROC SQL statements and a SAS DATA step. In addition, creating indexes on existing tables and normalization, or reducing data redundancy, are used to increase performance and optimize queries. Table relationships are established by defining primary and foreign keys. An integrated data dictionary is created to promote and enhance database management. Management of the database is provided for by queries, including the operations of select, project, and join. Data definition and data manipulation operations such as create, alter, insert, and update must also be performed to define and fine-tune the database.

INTRODUCTION
The purpose of this paper is to demonstrate the creation of a SQL database using the SAS System. The Structured Query Language or SQL (pronounced sequel in this paper) is implemented in Release 6.06 of the SAS System with PROC SQL. SQL is a powerful database language, yet it is easy to learn and understand because of its English-like syntax. For this reason it has become the ANSI standard for relational database languages. PROC SQL enables SAS users to define and alter a database, change or manipulate data within a database, and query or retrieve data from a database.

There are many relational database terms that may be unfamiliar to users of the SAS System. Following are definitions of some of the more common relational database terms:

- foreign key
  is a single column variable or a group of columns that is the primary key for another table.
- join
  is a query that joins rows (observations) in one table with rows in another table by equating common values in corresponding columns. For example, the column SSN may appear in the ADDRESS table and in the EMPLOYEE table. A PROC SQL query can select an employee’s last name from the EMPLOYEE table and his or her phone number from the ADDRESS table on the condition that the EMPLOYEE SSN equals the ADDRESS SSN. The resulting output displays only last names and phone numbers. No new data set is created, unless the join is contained in a CREATE TABLE statement.
- normalization
  is the process by which data redundancy is reduced to prevent update dependencies and conserve space.
- primary key
  is a single column variable or a group of columns containing data values that will always uniquely identify a single table row (observation). For example, Social Security number (SSN) may be the primary key value in a table containing employee names since a single SSN will always identify a single row.

query
is a SQL SELECT statement used to retrieve data from the database. Unlike the SAS DATA step, a query does not generate a new data set (unless included in a SQL CREATE TABLE statement). Rows retrieved are displayed in the SAS OUTPUT window.

table
is a data structure that holds data. In a relational database, a table is composed of rows and columns. In the SAS System, a table is a SAS data set, which is composed of variables and observations.

view
is a database object that is a logical representation of a table, group of tables, or any subset or superset thereof. Views appear to be identical to tables with the exception that in Release 6.06 views are read-only. Views do not contain data. Instead, they point to data contained in other tables. Views consume only the space required to store their definition. A view definition is any valid SQL query. They are defined with the CREATE VIEW statement. Views are used anywhere SAS data sets or tables are used, except they are read-only and cannot have indexes defined directly on them, although they take advantage of indexes defined on their underlying table or tables.

The term table can be assumed to refer to both tables and PROC SQL views, unless otherwise stated. In other words, a view is considered a special case of a table. The term view can be used to refer to two different entities in the SAS System: the PROC SQL view and the PROC ACCESS view descriptor. To avoid confusion in this paper, the term view will always refer to a PROC SQL view. PROC ACCESS view descriptors will be explicitly referred to as such, or shortened to the term view descriptor.

This paper is divided into three broad sections. The first section describes the development of the Personnel database, which was developed for this paper. It is a collection of related database objects that contain fictitious data. The second section describes the development of a data dictionary for the database. The data dictionary is an essential database management tool that is used to keep track of all database objects. The third section discusses and gives examples of managing the database and its data dictionary using SQL views.

DEVELOPING A SQL DATABASE
Tables, Views, and the Importance of Indexes
The following sections describe the design and implementation of a SQL database that contains fictitious personnel data. A personnel database was chosen because it is universally familiar and has the possibility of being adapted to real user needs. The database has been normalized to reduce, and where possible, eliminate redundant data. Primary keys have been defined. The primary keys are defined and maintained by creating unique indexes on a key column or columns. A unique index prevents duplicate data values from appearing in those columns. Primary and foreign keys, as defined in the introduction, are important because they define relationships between tables. These relationships are used when SQL programs
and queries are being developed. They are especially useful when joining tables and creating views. Indexes also serve two other purposes. If a unique index is defined on a single column, that column can have a maximum of one null or missing value. This is important because in Release 6.06 of the SAS System, the NOT NULL option is not implemented in the CREATE TABLE statement. Last, and perhaps most important, indexes improve the performance of PROC SQL queries, as well as other SAS procedures, especially those using the BY statement.

Views will be created as needed for the Personnel database. Views are most useful when they replicate complex queries that are frequently used. As a security measure, they can allow certain users only partial, read-only access to a table.

Creating Database Libraries

Permanent SQL tables, views, and indexes are stored in SAS data libraries. An index is always associated with a single table (not a view) and, by definition, is located in the same library as its associated table. Views, on the other hand, can reference many tables in any library. Views are limited only by operating system level protections as to where they may reside and which tables they may read.

In a multiuser database, it is very important to establish from the beginning where each user will store tables. In addition, the same SAS LIBREF should always be used to refer to any single library containing database tables. All database LIBREFs should then be created in a single autoexec. Having a standard set of LIBREFs guarantees that users will be able to share queries and views, as well as prevent users from implementing their own heterogeneous organization systems. Establishing a simple, well-organized library structure is the first step in creating a database.

The Personnel database is a multiuser database. In addition, the assumption has been made that queries and views may at some time need to be ported to other systems based on SQL. With this assumption in mind, the SAS libraries were created with the following rules:

1. One library contains all the Personnel tables (but not necessarily all the views).
2. One library contains all the data dictionary tables and views.
3. Some users may have a single library in which to create their Personnel database views.
4. Standard LIBREFs are derived from user logon IDs, with the exception of the data dictionary library, which will use the LIBREF SYSTEM. The LIBREF for the library in 1, above, is derived from the database administrator’s logon ID.
5. Standard LIBREFs, documented in the data dictionary, will be used in all database queries and view definitions.

These five rules create a well-organized library structure that can be easily managed. In addition, the use of logon IDs as LIBREFs allows for portability of SQL code between other systems based on SQL. Many products based on SQL, including ORACLE DB2; and SQL/DS, employ a two-level table naming convention, where the first level is the user’s logon ID. Therefore, using rule 4 above, a SAS LIBREF table_name could correspond to another vendor’s logon_ID.table_name.

Designing the Personal Database and the Entity-Relationship Model

The first step in designing a database is to become familiar with the data. Using pencil and paper, the data is assigned column names with a maximum length of eight characters. Column names should be descriptive of the data they represent. For example, the Personnel columns containing first names and last names are called FNAME and LNAME, respectively. Each column is assigned a data type, either number or character (dates are numbers in the SAS System). The length of each data type is decided by reviewing existing data and, if appropriate, adding a reasonable amount for future data.

For example, the longest first name of any current employee was seven characters; however, it was decided that a length of 15 would be a reasonable number to assign this column. Fortunately, this number is not set in stone and can be easily lengthened without having to create a new table, by using the SQL ALTER statement.

The column length can also be shortened, although it would be dangerous to do so since names that are longer than the new length would be truncated. Some data, such as Social Security numbers, are constant.

The next step is to group the data into tables. This should also be done on paper first so that tables and their relationships can be clearly seen. Table names, like column names, can have a maximum length of eight characters. Table names should also be descriptive of the data they contain. For example, the SALARY table contains salary data. If there are a large number of tables in your database, it is suggested that a set of table, view, Index and column naming conventions be established. Each table should be drawn, listing its columns and data types. Columns that are primary key constituents should be labeled. Draw tables containing foreign keys next to their parent table, and connect them with a line. The example in Figure 1 uses the SAS formats to designate column data types and uses the symbol PK to indicate a primary key column.

Figure 1 Example Table Diagram

Note that the column EDUCATE.EMPNO is a foreign key for the EMPLOYEE table. Diagramming the database tables and their relationships is known as creating an Entity-Relationship Model or an E-R Model (see Figure 2 at the end of this paper for an E-R Model of the Personnel database).

Normalizing the Database

The tables in Figure 1 represent a small segment of the Personnel database. They demonstrate the process of normalization and display both primary and foreign keys. Normalization occurred when it was decided that queries for education information would be submitted very infrequently. Therefore, the EDUCATE table was designed in order to prevent the data it contained from slowing down other queries to the EMPLOYEE table. Since the column

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>EDUCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNAME</td>
<td>EMPNO</td>
</tr>
<tr>
<td>M</td>
<td>EMPNO</td>
</tr>
<tr>
<td>LNAME</td>
<td>SCHOOL</td>
</tr>
<tr>
<td>EMPNO</td>
<td>$11.</td>
</tr>
<tr>
<td>POSITION</td>
<td>SCHOOL</td>
</tr>
<tr>
<td>LEVEL</td>
<td>DEGREE</td>
</tr>
<tr>
<td>STARTDAT</td>
<td>$9.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>EDUCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$15.</td>
<td>EMPNO</td>
</tr>
<tr>
<td>SCHOOL</td>
<td>$11.</td>
</tr>
<tr>
<td>SCHOOL</td>
<td>EMPNO</td>
</tr>
<tr>
<td>DEGREE</td>
<td>M</td>
</tr>
<tr>
<td>$9.</td>
<td>M</td>
</tr>
</tbody>
</table>

408
EMPNO is a primary key in the EMPLOYEE table, it can be used to locate any employee name. Therefore, the EMPNO column (11 characters) was used in place of the columns FNAMA, INI and LNAME (32 characters total) in the EDUCATE table. Thus, data redundancy was reduced. If this sounds confusing, consider the following PROC SQL query:

```sql
proc sql;
select EXP.FNAME, EXP.LNAME, EXP.SCHOOL
from EXPLORE AS EXP
left JOIN EDUCATE AS EDU
where EXP.EMPNO = EDU.EMPNO;
```

The query above outputs all employees’ first and last names and the school or schools they attended, without creating a new table or data set. Since some employees may have attended more than one school, the EDUCATE table has a composite primary key, consisting of the columns EMPNO and SCHOOL.

After all the database tables and their relationships have been diagrammed, it is time to begin creating the database.

Methods Used to Create the Database

The data for the Personnel database was entered into operating system files prior to the design phase (see “Creating Database Libraries” earlier in this paper). PROC SQL has the CREATE TABLE statement, which creates a table with zero observations. The SQL INSERT statement can be used to insert data into a table one row at a time. The SAS System provides a method for simultaneously creating and loading data sets with the DATA step. Since a data set and a table are synonymous in the SAS System, the DATA step was used to create and load the tables. In addition, all columns were given a format and, where necessary, an informat in an effort to make PROC SQL queries portable to other systems based on SQL. Assigning a permanent SAS format in the DATA step promotes portability of PROC SQL programs by eliminating the need to use a SAS FORMAT statement within SQL queries.

DEVELOPING A DATA DICTIONARY

The Data Dictionary Defined

The data dictionary, sometimes referred to as the system catalog, is a collection of tables and views that contain information concerning the contents of all other database objects. Strictly speaking, it is data about data. The four data dictionary tables created for the Personnel database, like all tables in the SAS System, were created and are maintained by the user.

The SAS output in Output 1 was produced with a query to the TABLES dictionary table, and displays information about the Personnel data dictionary tables.

<table>
<thead>
<tr>
<th>The SAS System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>TABLES</td>
</tr>
<tr>
<td>INDEXES</td>
</tr>
<tr>
<td>VIEWS</td>
</tr>
<tr>
<td>COLS</td>
</tr>
</tbody>
</table>

Output 1 Data Dictionary Tables

Using the Data Dictionary

The data dictionary is useful to a database administrator (DBA) as well as to database users, provided the information in the tables is kept up to date. Using these tables, along with a carefully planned database design, a DBA can easily survey any aspect of the database. For example, the TABLES dictionary table contains information such as table name, number of indexes, number of columns, and LIBREFs. The SAS libraries associated with the Personnel database were designed to allow only one LIBREF for each user. The SAS LIBREFs derive names from logon IDs. This allows the DBA to quickly and easily locate the tables and views of any user. Security is still maintained since all users are responsible for the protection of their own libraries. Users and SQL programmers will find the data dictionary an invaluable reference tool for learning about the database, creating new database objects, and writing queries that take advantage of existing tables, views, and indexes.

Designing and Creating a Data Dictionary

The data dictionary is designed, normalized, and created essentially the same way as the Personnel database (see “Developing a SQL Database” earlier in this paper) except, before beginning the design phase, it must be decided what data to include in the dictionary.

At least one table should be included for each of the three types of database objects: tables, indexes, and views. These three tables can be further broken down into smaller tables, depending on how much information is needed. For example, it was decided that the data dictionary for the Personnel database would contain detailed information on columns, including length, position, type, format, informat and so on. Therefore, a fourth table was created called COLS. This table separated column information from data specific to tables, such as LIBREF, engine, and location. Likewise, information about users, file protection, SAS catalogs, or SAS/ACCESS view descriptors may need a separate dictionary table if such information is pertinent.

A query to the Personnel data dictionary displays the four tables and corresponding column definitions shown in Output 2. These tables were chosen in order to keep the dictionary structure simple. They contain only pertinent information and attempt to minimize the redundancy of data between the tables.

See Figure 3 at the end of this paper for an E-R Model of the data dictionary tables.
Data dictionary tables may seem somewhat bulky, especially when in the COLS table. However, these tables are not bulky if they are properly subsetted and joined using PROC SQL views. Designing procedure Output 2 Data Dictionary Columns

As database objects are being created and modified, new entries must be made into the appropriate dictionary tables.

Data dictionary tables may seem somewhat bulky, especially when they contain character columns as long as the REMARKS column. Note that the VIEWS table contains a column called CODELOC. This column stores the location of a file that has the code that generated the view. Saving this code is not necessary, but it may be useful for creating future views and queries. In addition, if a view is dropped, it can be easily re-created by submitting the saved SAS code.

MANAGING YOUR DATABASE WITH SQL VIEWS

Developing a Set of Views

At this point, the database has gone through an extensive design phase and an equally extensive implementation phase. All data now exists in the database and is available to the user. However, looking at the database, it becomes evident that there is a need to group data in different tables with each other, as well as to create subsets and supersets of tables. Subsets and supersets can be achieved by using the SQL WHERE statement and SAS functions, and by varying the object items in the SELECT clause. Object items are column names or SQL expressions. See the SAS Guide to the SQL Procedure for more information. PROC SQL allows tables to share data with each other by joining the tables in a query. It is important to keep in mind that a SQL join query, unlike a SAS match-merge, does not create a new table or data set. This is usually desirable since the data is already stored in a table, and storing it again may waste system resources. However, if it is obvious that a particular query will need to be performed often, or a join or nested SELECT statement is complex, it may be necessary to have that query available permanently. This can be done by including the query in a CREATE TABLE statement (which creates a table that duplicates or derives data stored in other tables) or by creating a SQL view. A view, as defined in the introduction, looks at data stored in other tables. With this in mind, a set of permanent views should be created to increase the size and usability of the database (and data dictionary) without consuming more disk space. The following are examples of views that were created for the Personnel database and the data dictionary.

VADDRESS displays employees' address data with their permission. concatenates first and last names, uses the SAS ZIPNAME function to retrieve state names. It joins the ADDRESS and EMPLOYEE tables and subsets rows with the WHERE clause.

VVESTED displays all vested employees by calculating which employees have had at least five years of service. It is derived from the EMPLOYEE table.

VSAYEAR displays the salesman with the greatest dollar sales for the year. It joins the SALES and EMPLOYEE tables using an in-line view and uses the MAX function.

VMAN_EMP displays EMPLOYEE data for managers only. It subsets the EMPLOYEE table with the WHERE clause.

VSYSDF displays column definitions of all data dictionary tables. It subsets the COLS table, using ORDER BY TABLENAME.

VTABDEF displays table definitions of all tables. It subsets the TABLES table, using ORDER BY TABLENAME.

VCOLATT displays column attributes of all tables. It subsets the COLS table, using ORDER BY TABLENAME.

See Screen 1 for a SAS log that displays the results of a DESCRIBE statement on each of the above views.

The views listed in Screen 1 are all considered permanent; therefore, information about them is stored in the data dictionary tables. Note that the VIEWS table contains a column called CODELOC. This column stores the location of a file that has the code that generated the view. Saving this code is not necessary, but it may be useful for creating future views and queries. In addition, if a view is dropped, it can be easily re-created by submitting the saved SAS code.

CONCLUSION

The SAS PROC SQL database is now complete. Or, more accurately, it is temporarily complete. The nature of databases, by definition, is dynamic. New data is anticipated and will need to be added. When designing the database, considerations have been made for this anticipated new data. But there will always be unanticipated data that will require the addition of new tables and views, and may require the restructuring of some existing database objects as well. Having a well-planned database that includes an E-R Model and a data dictionary, and using the versatile Structured Query Language, will help to make changes straightforward and efficient.

For a copy of the SAS code that produces the Personnel database, contact the author.

SAS is a registered trademark of SAS Institute Inc., Cary NC, USA.

SQL, DB2, and SQL/DS are trademarks of International Business Machines Corporation.

ORACLE is a registered trademark of Oracle Corporation.
PK = primary key  $n.$ = character format, width of $n$  n.n = number format, scale of n.n

Figure 2  Entity-Relationship Model for the Personnel Database

PK = primary key  $n.$ = character format, width of $n$  n.n = number format, scale of n.n

Figure 3  Entity-Relationship Model for the Data Dictionary
Screen 1: DESCRIBE of SQL Views