ABSTRACT

This paper provides an overview of Structured Query Language (SQL) and its implementation in Version 6 of the SAS® System.

The History of SQL

The Relational Data Model, proposed by Codd (1970), represents data in tables. Structured Query Language (SQL) is a language used for accessing data stored in tables. The SAS data set concept blends very nicely with the concept of a table in the relational data model. Both have columns (variables) and rows (observations). SAS data sets are a little more liberal than true relational model tables because they allow duplicate rows and have an inherent ordering. Nevertheless, they are similar enough to make SQL a useful language for accessing SAS data sets. The terms data base table and SAS data set are interchangeable in the context of this paper.

There are many commercial products that support SQL. Early SQL-based systems were written for mainframes and minicomputers. Recently, there have been several announcements of SQL-based products for microcomputers.

Structured Query Language

Implementations of SQL usually have two components:

- a set-at-a-time nonprocedural component, enabling a user to query and modify data base tables
- a record-at-a-time procedural component, usually embedded into third generation programming languages.

This paper is concerned with the set-at-a-time nonprocedural component and its implementation in the SAS System.

Advantages of Using SQL

SQL is a nonprocedural language. This is advantageous because the user does not have to be concerned with the details of actually processing the request. In short, the user specifies what is wanted and allows the application program to resolve the details of how to achieve the results. In addition, SQL syntax is similar to English. This makes SQL easier to understand and learn.

SQL has been implemented by many people and on many hardware platforms. Many new data bases offer a form of SQL, and vendors are retrofitting relational-query capabilities to existing products. Distributed data base systems are becoming viable. There are even products available that connect heterogeneous data bases to one another using SQL as the common thread. If the volume of articles on SQL in the popular computing press is any indication, many products that support SQL are in the works.

SQL vendors are actively pursuing a standardization of the language. Already, the SQL ANSI standard (ANSI 1986) has specified the basic building blocks of SQL. The ANSI-X3H2 technical committee is working on an updated standard (ANSI 1987) containing more features and addressing noted deficiencies in the language.
The SELECT Statement

The SELECT statement is used to query a table. In its simplest form (used above to display the sample data sets), the SELECT statement can be separated into clauses. The keyword SELECT introduces the object clause and the variables desired. The "*" is shorthand for all variables. A subset of the variables for a table may be requested also.

The keyword FROM introduces the table. The WHERE clause of the SELECT statement is used to specify the rows of a table to be displayed.

```sql
SELECT author, section, time
FROM sugLpaper
WHERE time>'12:00';
```

So far, SOL provides no more functionality than traditional SAS tools. However, SOL permits arbitrary expressions where variables might be specified. Suppose that the conference convenors decide to delay the papers for 30 minutes and want to display the new paper times. A single SOL statement achieves the same result that would have required three SAS steps (a DATA step to create the new variable, a SORT procedure, and a PRINT procedure).

```sql
SELECT author, section, title,
time+'0:30't as newtime format=time5.
FROM sugi .paper
ORDER by section, time;
```

You can use all the functions available to the DATA step in SOL expressions. SAS Institute supplies many more functions than required by the SOL standard. In the example below, a function call is used in the WHERE clause to subset the data retrieved, without displaying the results of that function call.

```sql
SELECT author, section, title
FROM sugi .paper
WHERE scan(Section, 2)='Sys' or
'M' =substr(authOr, 1, 1);
```

SQL Features for Summary Statistics

SQL provides summary (or aggregation) operators. Any or all of the following statistics can be requested for the entire table, or on a per group basis: MIN, MAX, COUNT, SUM, AVG, SUMWGT, SS, CSS, VAR, STD.

```sql
SELECT max(rating) as maxr, min(rating) as minr
FROM sugi.attend;
```

If you wanted the statistic by section, rather than for the entire table, you would have to look up the section names using the SUGI.PAPER table, matching rows on author name.

```sql
SELECT paper.section, max(rating) as maxr, min(rating) as minr
FROM sugi.attend, sugi.paper
WHERE attend.author=paper.author
GROUP by paper.section;
```

SQL HAVING clauses can be considered WHERE clauses for each group of a query involving summary statistics and may reference both elementary data items as well as summary functions. (This feature is not available in many SOL implementations whose workaround is similar to the traditional SAS solution.) You must create a table with the maxima, and merge those values back with the original data.

```sql
SELECT paper.author, paper.section, rating
FROM sugi.attend, sugi.paper
WHERE attend.author=paper.author
GROUP by paper.section
HAVING rating=max(rating);
```

Warning: The query as specified involves remerging the summary statistics back with the data that creates those statistics. This may not be what you had intended!

Multiple Table Queries

So far, PROC SOL, with its nonprocedural SOL syntax, has provided some improvements over traditional procedural solutions to problems. But there is more! SOL deals with multiple input tables in an intuitive fashion. The user is free to concentrate on
what data are desired while the system concerns itself with how
the data are produced.

At the hypothetical conference used in this example, all papers
in a section are given in the same room. When you want to print
the program, you must obtain the room information from another
table.

SQL makes this quite simple. You can join any number of tables
by listing more than one in the FROM clause of the query. If you
want to achieve some kind of matching between the rows of the
various tables, you must specify this in the WHERE clause. These
row matching conditions are often called join predicates.

You can join more than two tables in any single query. Recall that
the hotel management has provided the capacity of the rooms
used (SUGI.CAPACITY), and conference staffers estimated the
attendance of papers (SUGI.ATTEND).

Unfortunately, conference staffers did not record the room or the
section. All you had were scraps of paper with the author, an esti­
mated number of people in the audience, and a rating of the
audience reaction to the paper on a scale of 1 to 5.

You would like to see the room-utilization data by paper. This
involves four tables! First, you must get the attendance details
from the SUGI.ATTEND table. To get the section details, you
must get the capacity from SUGI.CAPACITY.


create view prizes

<table>
<thead>
<tr>
<th>AUTHOR</th>
<th>AWARD</th>
<th>SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marti</td>
<td>SUGI pen</td>
<td>Graphics</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI T-shirt</td>
<td>Graphics</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI steak knives</td>
<td>Graphics</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI pen</td>
<td>Info Sys</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI T-shirt</td>
<td>Users</td>
</tr>
</tbody>
</table>

As far as the user is concerned, views and tables are inter­
changeable. You can restrict the rows displayed from a view
using the same WHERE clause syntax as before. You can join
views with other views or with base tables. Views can reference
other views!


create view prizes2

<table>
<thead>
<tr>
<th>AUTHOR</th>
<th>AWARD</th>
<th>SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marti</td>
<td>SUGI pen</td>
<td>Graphics</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI T-shirt</td>
<td>Graphics</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI steak knives</td>
<td>Graphics</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI pen</td>
<td>Info Sys</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI T-shirt</td>
<td>Info Sys</td>
</tr>
<tr>
<td>Tom</td>
<td>SUGI pen</td>
<td>Testing</td>
</tr>
<tr>
<td>Tom</td>
<td>SUGI T-shirt</td>
<td>Testing</td>
</tr>
</tbody>
</table>

Another property of SQL joins is that the match condition need
not necessarily be an equals match. At the hypothetical confer­
cence, prizes are handed out based on the rating given to the
presenter. The awards are cumulative. If a presenter receives a
rating of four, the presenter qualifies for the award for a four, as
well as all lesser rated prizes.

Another property of SQL is that the condition need not
necessarily be an equals match. At the hypothetical confer­
cence, prizes are handed out based on the rating given to the
presenter. The awards are cumulative. If a presenter receives a
rating of four, the presenter qualifies for the award for a four, as
well as all lesser rated prizes.

```
select * from prizes
where author='Marti';
```

```
select * from prizes2
where prizes.author=author;
```

```
create view prizes3

<table>
<thead>
<tr>
<th>AUTHOR</th>
<th>AWARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marti</td>
<td>SUGI pen</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI T-shirt</td>
</tr>
<tr>
<td>Marti</td>
<td>SUGI steak knives</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI pen</td>
</tr>
<tr>
<td>Lewis</td>
<td>SUGI T-shirt</td>
</tr>
<tr>
<td>Tom</td>
<td>SUGI pen</td>
</tr>
<tr>
<td>Tom</td>
<td>SUGI T-shirt</td>
</tr>
</tbody>
</table>

```

Subqueries in SQL

Sometimes, you may not know the value of the variable to be
used in your selection criteria, or it may vary from row to row for
the table being processed. For example, "Whose papers are in
the section convened by Denise?"
PROC SQL supports correlated subqueries too. A correlated subquery is an inner query that cannot be evaluated without referring to the current value of some variable in the outer query. Date (1981) gives examples on correlated subqueries.

Data Manipulation in SQL

So far, retrieving values from a database has been discussed. SQL also supports INSERT, DELETE, and UPDATE statements. You can insert constant values or the result of a query expression into a table. An example might be

```
insert into high .. Uy
    select *
    from employee having rating>.9.maxrating;
```

The DELETE statement allows you to qualify the records that you would like to remove.

```
delete * from payroll where status='Fired';
```

The UPDATE statement allows "in-place" updating of a SAS data set.

```
update payroll
    set salary=.1*salary,
        bonus=.9*bonus
    where dept='Sales';
```

SQL also has a CREATE and DROP statement. As you have seen, you can use the CREATE VIEW statement to define views. There are also CREATE TABLE and CREATE INDEX statements. You might use these over the functionally equivalent DATA step or PROC DATASETS if you already had the table definition from another SQL-based application or if you were more familiar with SQL than the SAS language.

The DROP statement will drop tables, views, and indexes.

Implementation within the SAS System

SAS Institute will distribute a beta test version of SQL with the next release of the SAS System. It will include PROC SQL, enabling you to enter SQL statements to execute queries, define tables, views, and indexes.

Views defined with PROC SQL will be accessible as read-only tables to all SAS procedures as if they were data sets. Recall the prizes view that selected data from more than one table. When looking for datasets, the SAS System will look for a view if no real table exists with the given name. If a view exists, then the procedure will process the view as if it were a real table. The SQL view engine that enables SAS procedures to process views as base tables will also be considered in beta test.

```
proc chart data=view.prizes;
    hbar award;
    run;
```

WARNING: The value of AWARD will be truncated to 16 characters.

FREQUENCY OF AWARD

<table>
<thead>
<tr>
<th>AWARD</th>
<th>FREQUENCY</th>
<th>PERCENT</th>
<th>PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUGI T-shirt</td>
<td>**********</td>
<td>4</td>
<td>40.00</td>
</tr>
<tr>
<td>SUGI pen</td>
<td>***********</td>
<td>5</td>
<td>50.00</td>
</tr>
<tr>
<td>SUGI steak knife</td>
<td>*****</td>
<td>1</td>
<td>10.00</td>
</tr>
</tbody>
</table>

FREQUENCY

Plans for the Future

When SQL is integrated within the multiple-engine architecture of the SAS System, you will be able to define SQL views that span heterogeneous data sources. This will make the task of presenting production level data to end-users easier for MIS people.

The Institute plans to develop the embedded SQL syntax and make it available in SAS programming languages such as the DATA step and Screen Control Language (SCL) of SAS/AF® software.

Also, the Institute plans to allow updating of views that are defined on simple base tables. You will not be able to update derived fields or fields derived from summary statistic calculations.

The updating of views defined over more than one table presents many technical challenges. Part of the ongoing research into this area may result in the capability to update these views as if they were base tables.

REFERENCES


SAS and SAS/AF are registered trademarks of SAS Institute Inc., Cary, NC.