

A Taste of SQL

Paul Kent
SAS Institute Inc., Cary, NC
<kent@unx.sas.com>

embedded into third generation programming languages. Support for Embedded SQL may be added in future versions of the SAS System.

Abstract

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

Advantages of using SQL

SQL is a non-procedural language. An advantage of this is that the user does not have to concern himself with the details of actually processing the request. In short one gets to say **WHAT** they want, and allow the application program to resolve the nitty-gritty details of **HOW** to get the results. In addition SQL syntax is "English-like" promoting quick learning of its constructs.

SQL has been touted as providing data independence. This is not as big a selling point to SAS folks — a SAS data set has always protected applications from changes to the underlying files and most carefully written programs are immune to new variables in the data sets they process.

SQL has been implemented by many people, on many hardware platforms. Many new database solutions offer a form of SQL, and existing vendors are retro fitting relational-query capabilities to their products. Distributed database systems are becoming viable — there are even products available that connect heterogeneous databases using SQL as the common thread. By the volume of SQL articles in the popular computing press, many people are working on products that support it.

The SQL vendors are actively pursuing a standardisation of the language. The SQL ANSI standard (ANSI86) already specifies the basic building blocks of SQL, and the ANSI-X3H2 technical committee is at work on an updated standard (ANSI87) that contains more features, and addresses noted deficiencies in the language.

SQL is the database access language of IBM's SAA (IBM87). Systems Application Architecture is the IBM blueprint for creating portable programs that can run on all IBM hardware.

The History of SQL

The "Relational Data Model", proposed by Codd (CODD70) represents data 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 — they allow duplicate rows, and have an inherent ordering. Nevertheless, there are strong enough parallels between the two to make SQL a useful language for accessing SAS data sets. The terms database table and SAS data set are interchangeable in the context of this paper.

SQL is a language for accessing and manipulating data stored in tables. SQL is an acronym for "Structured Query Language" and is pronounced "ess-que-ell" or "sequel".

There are many commercial products that support SQL. Early SQL-based systems were written for mainframes and minicomputers. Recently there have been many news items announcing SQL-based products for microcomputers — one can hardly read a trade rag without encountering "client/server".

Structured Query Language

Implementations of SQL usually have two components for manipulating the data stored in the data base.

- A set-at-a-time non-procedural component, allowing a user to query and modify database tables. This paper is concerned with this component, and its implementation in the SAS System.
- A record-at-a-time procedural component, that is usually

Advantages of SQL for a SAS user

PROC SQL provides an alternative to existing SAS solutions to many data processing problems. The key advantage of SQL over traditional SAS solutions stem from the nonprocedural language — SQL solutions do not require lots of procedural framework.

PROC SQL solutions are “self optimising”. SQL will take advantage of indexes and inherent sort order in your data sets. If you add a useful index at some later point, all your SQL programs will take advantage of it. Most of your procedural programs will need re-writing to take advantage of the newly added index.

PROC SQL allows more direct communication to other SQL data bases via SAS/ACCESS Software. The SQL Passthru facility is often the most efficient way to extract information from an external DBMS.

PROC SQL may allow you to transfer the applications logic from some database application directly into SAS. This is useful if you were already using SAS for the reporting phases of the application.

SQL Statements

There are six main statements in the non-procedural component of SQL.

SELECT	to retrieve values from database tables the meet a user's specification.
INSERT	to insert rows into database tables.
DELETE	to delete rows from database tables. Generally a DELETE statement is qualified with an expression as to which rows to delete.
UPDATE	to modify the values of rows in database tables.
CREATE	to create database tables, views and indexes.
DROP	to remove database tables, views and indexes.

The example database

The examples in this paper are based on a set of tables recording things about events at a users group meeting. The tables are all stored in a SAS data library that has a libname of SUGI, and reproduced here for reference.

These example data sets are an abridged version of those in the SUGI13 paper “SQL and the SAS System”. These same sample data sets are in the SAS SAMPLE LIBRARY — look for members starting with SQL. The examples are also in these members, and display more variations than shown in this paper.

```
proc sql;
  /** PAPERS PRESENTED **/
  select * from sugi.paper;
```

AUTHOR	SECTION	TITLE	..	TIME
Paul	Info Sys	Query L	..	10:30
Jim	Users	Startin	..	11:15
Marti	Graphics	Multi-d	..	14:30
Lewis	Info Sys	Query O	..	15:30
Tom	Testing	Automat	..	9:00
Jane	Graphics	Making	..	16:15

```
  /** SECTION CONVENORS **/
  /** ROOMS ALLOCATED **/
  select * from sugi.section;
```

SECTION	ROOM	CONVENOR
Graphics	Sable	Denise
Info Sys	Kudu	Peter
Testing	Sable	Linda
Users	Kudu	Fred

```
  /** ROOM CAPACITIES **/
  select * from sugi.capacity;
```

ROOM	CAPACITY
Kudu	150
Sable	200

During the conference, the authors presentations were judged and the number of attendees at each paper were estimated. This data is recorded in a table too.

It is customary to present awards to the speakers based on their ratings. These awards are also recorded in a table.

```

/** ATTENDANCE FIGURES **/
/** AND RATINGS **/
select * from sugi.attend;

```

AUTHOR	RATING	ATTEND
Paul	3	75
Jim	4	125
Marti	5	180
Lewis	4	55
Tom	4	105
Jane	2	160

```

/** AWARDS FOR PRESENTERS **/
select * from sugi.awards;

```

RATING	AWARD
3	SUGI pen
4	SUGI T-shirt
5	SUGI steak knives

So far, SQL provides no added functionality over traditional SAS tools. However, SQL permits arbitrary expressions where variables might be specified. Suppose that the conference convenors decide to delay the papers for 30 minutes, and wish to display the new paper times. A single SQL statement achieves the same result that would have required three SAS steps. (A DATA STEP to create the new variable, then a PROC SORT and finally a PROC PRINT).

```

select author, section, title,
       time + '0:30't
       as newtime format=time5.
from sugi.paper
order by section, time;

```

AUTHOR	SECTION	TITLE	..	NEWTIME
Marti	Graphics	Multi-	..	15:00
Jane	Graphics	Making	..	16:45
Paul	Info Sys	Query	..	11:00
Lewis	Info Sys	Query	..	16:00
Tom	Testing	Automa	..	9:30
Jim	Users	Starti	..	11:45

The SELECT statement

The SELECT statement is used to query a table. In its simplest form (used above to display the sample data sets), it can be separated into clauses.

The keyword SELECT introduces the object clause and lists the variables that you desire. The * is short-hand for all variables.

The keyword FROM introduces the table that you are interested in.

The WHERE clause of the SELECT statement is used to specify which rows of a table that you want to process.

```

select author, section, time
from sugi.paper
where time > '12:00't;

```

AUTHOR	SECTION	TIME
Marti	Graphics	14:30
Lewis	Info Sys	15:30
Jane	Graphics	16:15

You can use all the functions available to the DATA STEP in SQL expressions. In this example, we use the SCAN and SUBSTR functions in the where clause. Notice that you need not necessarily display variables used in selecting the rows. SAS Institute supplies many more functions than required by the SQL standard, and you can supply your own user-written functions with SAS/TOOLKIT Software.

```

select author, section, title
from sugi.paper
where scan(section, 2) = 'Sys'
or substr(author,1,1) = 'M'

```

AUTHOR	SECTION	TITLE
Paul	Info Sys	Query L ..
Marti	Graphics	Multi-d ..
Lewis	Info Sys	Query O ..

SQL features for summary statistics

SQL provides summary (or aggregation) operators. You can request any or all of the following statistics, for the entire table, or on a per group basis:

MIN, MAX, COUNT, SUM, AVG, SUMWGT, SS, CSS, VAR, STD

```
select max(rating) as maxr,
       min(rating) as minr
from sugi.attend;
```

MAXR	MINR
5	2

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. (Author name is the only link to section name in the tables we have been given) The traditional SAS solution would require SORTing and MERGEing the attend and paper data sets, followed by a SUMMARY.

```
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;
```

SECTION	MAXR	MINR
Graphics	5	2
Info Sys	4	3
Testing	4	4
Users	4	4

PROC SQL also implements the ability to reference the elementary data items as well as the summary statistics in the same expression. This process of remerging the statistics back together with the data that generated them is useful for answering questions like 'Who earned the most in each division?' The traditional SAS solution for a problem like this would involve creating a summary data set with the maximum for each department, then merging that data set with the original data looking for records with the calculated maxima.

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 SQL implementations — whose work around is similar to the traditional SAS solution — create a table with the maxima, and merge those values back with the original data.

```
select paper.author,
       paper.section,
       rating
from sugi.attend, sugi.paper
where attend.author = paper.author
group by paper.section
having rating = max(rating);
```

AUTHOR	SECTION	RATING
Marti	Graphics	5
Lewis	Info Sys	4
Tom	Testing	4
Jim	Users	4

Multiple table queries

So far, PROC SQL with its non procedural SQL syntax has provided some improvements over traditional procedural solutions to problems. But there is more! SQL deals with multiple input tables in an intuitive fashion — the user is free to concentrate on the WHAT, while the system concerns itself with the HOW.

At our hypothetical conference, all papers in a section are given in the same room. When we wish to print the program, we 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 on the FROM clause of the query. If you want to achieve some kind of matching between the rows of the various tables, you specify this in the where clause. These row matching conditions are often called join predicates.

```
select time, paper.section,
       room, author, title
from sugi.paper, sugi.section
```

```
where paper.section
      = section.section
order by time;
```

TIME	SECTION	ROOM	AUTHOR	TITLE..
9:00	Testing	Sable	Tom	Autom..
10:30	Info Sys	Kudu	Paul	Query..
11:15	Users	Kudu	Jim	Start..
14:30	Graphics	Sable	Marti	Multi..
15:30	Info Sys	Kudu	Lewis	Query..
16:15	Graphics	Sable	Jane	Makin..

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

Unfortunately they did not record the room or the section — all we had were scraps of paper with the author and an estimate of the number of people in the audience, and a rating of the audience reaction to the paper on a scale of 1 to 5.

We would like to see the room-utilisation data by paper. This involves four tables! First, we get the attendance details from the attend table. To get the section details we will need to access SUGI.PAPER, cross referencing author names and their sections. Once we have the sections, we can get the room from the SUGI.SECTION table by cross referencing on the section variable. Now that we have the room, we can pick up the room capacity from SUGI.CAPACITY and voila!

Of course, we should have designed our tables correctly at the outset, but in real-world situations one must often make do with the information that is available. SQL makes following the threads that link diverse data tables together a little easier.

```
select attend.author, title,
       capacity.room,
       (attend/capacity)*100
       as utilised
from sugi.attend A,
     sugi.paper P,
     sugi.section S,
     sugi.capacity C
where A.author = P.author
     and P.section = S.section
```

```
and S.room = C.room;
```

AUTHOR	TITL	.. ROOM	UTILISED
Paul	Quer	.. Kudu	50
Lewis	Quer	.. Kudu	36.66667
Jim	Star	.. Kudu	83.33333
Marti	Mult	.. Sable	90
Jane	Maki	.. Sable	80
Tom	Auto	.. Sable	52.5

Another property of SQL joins is that the match condition need not necessarily be an "equals" match. At our hypothetical conference we hand out prizes based on the rating given to the presenter. And that's not all — the awards are cumulative, so if you get a rating of 4 you can expect two wonderful objects d'art!

```
select author, award
from sugi.attend A1,
     sugi.awards A2
where A1.rating >= A2.rating
order by author;
```

AUTHOR	AWARD
Jim	SUGI pen
Jim	SUGI T-shirt
Lewis	SUGI pen
Lewis	SUGI T-shirt
Marti	SUGI pen
Marti	SUGI T-shirt
Marti	SUGI steak knives
Paul	SUGI pen
Tom	SUGI pen
Tom	SUGI T-shirt

SQL Views

Often, you would like to define subsets of the total database as user-views of the data. SQL provides this capacity through stored views. You can store any select statement as a view, and subsequently retrieve the data through the view name.

```
create view prizes
as
select author, award
  from sugi.attend A1,
      sugi.awards A2
 where A1.rating >= A2.rating
 order by author;
```

Note: View USER.PRIZES has been output.

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

AUTHOR	AWARD
Marti	SUGI pen
Marti	SUGI T-shirt
Marti	SUGI steak knives

As far as the user is concerned, views and tables are interchangeable. 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
as
select prizes.author,
       award, section
  from prizes P1,
      sugi.paper P2
 where P1.author = P2.author;

select * from prizes2;
```

AUTHOR	AWARD	SECTION
Paul	SUGI pen	Info Sys
Jim	SUGI pen	Users
Jim	SUGI T-shirt	Users
Marti	SUGI pen	Graphics
Marti	SUGI T-shirt	Graphics
Marti	SUGI steak knives	Graphics
Lewis	SUGI pen	Info Sys
Lewis	SUGI T-shirt	Info Sys
Tom	SUGI pen	Testing
Tom	SUGI T-shirt	Testing

SUBQUERIES in SQL

Sometimes, you don't 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?"

```
select author
  from sugi.paper
 where section =
    ( select section
      from sugi.section
    where convenor = 'Denise'
    );
```

AUTHOR
Marti
Jane

PROC SQL supports correlated subqueries too. A correlated subquery is one where the inner query cannot be evaluated without referring to the current value of some variable in the outer query. Chris Date, in his book "An Introduction to Database Systems", gives examples on correlated subqueries.

Data manipulation in SQL

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

```
insert into high_fly
select *
  from employee
 having rating > .9*max(rating);
```

The DELETE statement allows you to qualify which 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 = 1.10*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 you were more familiar with SQL than the SAS language.

The DROP statement will drop tables, views and indexes.

New Features in PROC SQL for Release 6.07

We concentrated our efforts in three main areas: Performance, DICTIONARY TABLES and Support for External SQL.

The performance of the SAS System in general has improved with Release 6.07 and PROC SQL benefits from this. We have also

- enhanced our code to recognise the SORT Order information stored in SAS data sets to avoid internal sorting phases
- added code to perform some joins as an in-memory join using a hashing technique to identify rows that match — this avoids sorting at the cost of using more memory

DICTIONARY Tables are "pseudo-tables" that PROC SQL materialises on demand. They contain information about the context of the SAS execution. An example usage that computes the number of observations that fit in a buffer for all tables in the SASHELP library whose name begins with 'A' would be:

```
select memname, obslen,
       bufsize, nob,
       floor(bufsize/obslen)
       as bufobs
  from dictionary.tables
 where libname = 'SASHELP'
       and memname like 'A%';
```

Member	Observation	..
Name	Length	Bufsi..
ADBEEX	80	40..
ADBLOC	119	40..
ADDON	72	40..
ADXPARM	236	40..

Supporting External SQL is called the "Pass-Through" in the BASE SAS Changes and Enhancements Documentation(SAS Technical Report p-222). We have added syntax that allows you to send SQL commands directly to the underlying database. An example of this new syntax that creates a table in the database, inserts a row, and then retrieves that row is:

```
PROC SQL;

EXECUTE (
  create table test
    ( a int, b int )
)
BY DB2;

EXECUTE (
  insert into test
    values(1,2)
)
BY DB2;

select *
  from CONNECTION TO DB2
    ( select * from test )
;
```

References

(ANSI86) X3.135 "Database Language SQL".

(ANSI87) X3H2-87-303 "Working draft SQL2", December 1987.

(CODD70) Codd, E.F. "A relational model of data for large shared data banks", CACM 13 #6, June 1970.

(DATE81) Date, C.J. "An Introduction to Database Systems, Volume 1", Addison-Wesley, 1981. ISBN 0-201-51381-1

(IBM87) IBM "Systems Application Architecture — Common Programming Interface Database Reference", IBM SC26-4348-0, September 1987.

"Database Programming & Design", Miller Freeman Publications, ISSN 0895-4518, a monthly publication

"DBMS", M&T Publishing, ISSN 1041-5173, a monthly publication

"SAS Users Group International Conference Proceedings", 1988 through 1992 have papers that reference SQL