Abstract:
SQL is an ANSI standard database query language. Its syntax is simple, and its utility is broad in scope. SAS® provides a SQL programming environment via proc SQL. This presentation will elaborate on SQL's major functions, using proc SQL syntax.

Introduction:
SQL, an acronym for Structured Query Language, is an ANSI (American National Standards Institute) standard database programming/query language. Although its syntax is relatively compact and simple, it provides a great deal of utility. Its combination of simplicity, power and usefulness has contributed to its ubiquity. With a cursory knowledge of basic SQL syntax, one is able to create, view, combine and manage data stored in variety of database management systems. The purpose of this paper is to present to the uninitiated SQL user, the proc SQL syntax necessary to perform a wide range of basic querying and data management functions. The examples presented herein will demonstrate how to use proc SQL to view data, create and manage SAS data objects, summarize and combine data, plus enhance query performance.

Proc SQL Procedure Basics:
To start the SQL procedure, you must issue the following code:

```
proc sql;
```

Once the procedure has been started, you may submit one or more SQL statements for immediate processing. A SQL statement is a string of SQL code that ends in a semi-colon. A syntactically correct statement, submitted in full (i.e. ending in a semi-colon) is processed immediately. You do not need to submit 'run;' to execute any SQL statements. Data generated or modified by a SQL statement are immediately available to the very next SQL statement.

To end the proc SQL instance, submit the following code.

```
quit;
```

All code examples that follow are written as if an instance of proc SQL already exists. If an instance does not exist, one can be started by submitting, 'proc SQL;'.

Viewing SAS Data:
Generally, the SELECT-FROMSQL statement, on its own, will write data from the selected columns in the designated table, to the output window or listing. The basic SELECT-FROM syntax is listed below.

```
SELECT col umn- name <,column name>
FROM tab le- name;
```

When you select individual columns from a table, you must comma-delimit your SELECT statement's column name list. The shortcut to select all columns from a table is to write an asterisk, following the SELECT statement.

In the example below, all columns from the table SASHELP.CLASS will be written to the SAS output window or listing. Note that proc SQL's print option is being activated by a RESET statement.

```
reset print;
select *
from sashelp.class;
```

Use a WHERE clause to define a filter to restrict which records are processed by your query. In the example below, only girls' names are displayed. The UPCASE function is used to ensure that both lowercase and uppercase values of 'f' pass through the filter.

```
select name
from sashelp.class
where upcase(sex)='F';
```

If you wish to include row numbers in your output, prefix your SELECT-FROM code with the following proc SQL option statement.

```
reset number;
```

If you wish to restrict the number of rows proc SQL processes from any single data source, set the INOBS= option to the desired number of rows. The following option setting limits the number of records read from any given source table to 10.

```
reset inobs=10;
```

To remove any restriction set by the INOBS= option, submit the following code.

```
reset inobs=MAX;
```

The INOBS option is useful for trouble shooting queries that produce many records of output.
**Describing a SAS Table or View:**

Proc SQL's `DESCRIBE TABLE` statement writes, to the SAS log, structural information on the requested table. The statement's syntax is listed below.

```
DESCRIBE TABLE table-name;
```

Note that the `DESCRIBE TABLE` statement's output is written as a valid SQL CREATE TABLE statement. This feature's output can be used to recreate the described table in SAS or another relational database system. A word of warning: the recreated table would have 0 observations.

A SQL view is a stored query whose instructions are processed when the view is executed. A view contains no actual data, however its execution may lead to the creation of data. Proc SQL's `DESCRIBE VIEW` statement writes to the SAS log the SQL view's definition in the form of a valid SELECT-FROM statement. This feature allows the user to see how the described view was defined. The statement's syntax is listed below.

```
DESCRIBE VIEW view-name;
```

**Creating a SAS Table or View:**

There are three ways in which you can create a table using proc SQL, all of which use the CREATE TABLE statement. The first method creates a blank table with columns and their assigned data types as they are explicitly defined. This method's syntax, shown below, is the same returned by DESCRIBE TABLE.

```
CREATE TABLE table-name (column-specification <, column-specification> ...
<, constraint-specification> ...);
```

The code below creates a blank table in the WORK library called CLASS.

```
CREATE WORK.TABLE CLASS
(name char(8) not null,
sex char(1),
age num,
height num,
weight num,
constraint sex_MF
check(sex in('M','F')));
```

The newly created table, WORK.CLASS, has 5 columns. The columns NAME and SEX are character type and have lengths of 8 and 1 byte, respectively. The remaining three columns, AGE, HEIGHT and WEIGHT are numeric type and by default store 8 bytes each. Integrity constraints have been defined for the columns NAME and SEX. Both methods for defining an integrity constraint, written in italicized text, have been used. Because of these integrity constraints, no row of data can be inserted if the NAME field is null, or the SEX field contains any value other than 'M' or 'F'. Other integrity constraints have been available since the release of SAS V8, and it's recommended that you explore further this relatively new (to SAS), but highly useful feature.

You can create a blank clone of another table by using the CREATE TABLE statement with the LIKE clause.

```
CREATE TABLE table-name LIKE table-name;
```

Finally, you can create a table via a query expression.

```
CREATE TABLE table-name AS query-expression
<ORDER BY order-by-item <,order-by-item>...>;
```

The result of your query expression is written to a SAS table, specified in table-name. In the following example, a table WORK.MALES is created by selecting all columns from SASHELP.CLASS, and only those records where the column, SEX, has the value 'M'. The resulting table is sorted in descending order by weight. In the ORDER BY statement, the keyword DESC, causes the sort to be in descending order for the column(s) after which it immediately follows. By default the ORDER BY statement sorts data in ascending order.

```
cREATE TABLE males as
select *
from sashelp.class
where sex='M'
order by weight desc;
```

SAS SQL views can only be created via a query expression.

```
CREATE VIEW view-name AS query-expression
<ORDER BY order-by-item <,order-by-item>...>;
```

Based on the following example, the view, TALLEST_FEM, when executed, will produce a single row containing the greatest value of HEIGHT for those records whose column SEX = 'F'. Note that we are using the MAX() function to determine what value of HEIGHT is the greatest from among all the rows that have the column SEX = 'F'. Also note that we are storing the result of our summary function into a new column called MAX_F_HEIGHT.

```
create table males as
select *
from sashelp.class
where sex='M'
order by weight desc;
```
create view tallest_fem as
select max(height) as max_f_height
from sashelp.class
where sex='F';

Dropping SAS Tables and Views:
You can use proc SQL to delete or drop tables and views. The syntax for dropping tables and views is listed below.

DROP TABLE table-name <,table-name>
...

DROP VIEW viewname <,viewname> ...;

The following proc SQL statement deletes the tables TEMP1 and TEMP2 from the WORK library:
drop table temp1, temp2;

Summarizing Data:
Stratification for data summarization is defined in SQL’s GROUP BY statement. Proc SQL can produce a table or listing that contains summary level information for each unique value within a column or combination of values across columns listed in its GROUP BY statement. For instance, the following query lists, as SAS output, totals for the number of boys and girls in the table SASHELP.CLASS:

create table height_sum as
select sex,
    count(*) as count,
    avg(height) as avg_height,
    min(height) as min_height,
    max(height) as max_height,
    std(height) as std_height
from sashelp.class
group by sex,
    age
having avg(height) > 60;

Combining Data:
There are many ways to combine data from two or more data sources. This paper will demonstrate how to control how data sources are combined using various SQL joins. There are two principle types of joins: inner and outer (full, right and left). An inner join produces a result set that contain records from all source tables that share common key column(s) values. The syntax to perform an inner join is listed below.

create table height_sum as
select sex,
    age,
    count(*) as count,
    avg(height) as avg_height,
    min(height) as min_height,
    max(height) as max_height,
    std(height) as std_height
from sashelp.class
group by sex,
    age
having avg(height) > 60;

There are two principle types of joins: inner and outer (full, right and left). An inner join produces a result set that contain records from all source tables that share common key column(s) values. The syntax to perform an inner join is listed below.

select a.index,
    a.field1,
    b.field2
from one a
inner join
    two b
on a.index=b.index;

Because SEX appears in the GROUP BY statement, the summary function, COUNT(*), produces a total count for each unique value of SEX (in our example, assume the column SEX contains only two possible values: 'M' and 'F'). This query will yield two columns, with two records: one record containing the total number of boys and another with the total number of girls. The most common sex will be listed first, because the ORDER BY statement sorts our results in descending order by the calculated field, COUNT.

Generally, only those fields listed in the GROUP BY statement, plus any summarized fields are listed in the SELECT statement. For some SQL (DB2 & Oracle) this is a rule, however SAS’s proc SQL does not impose the said restriction. It’s recommended that you follow this rule, until you become more familiar with how SQL summarizes data.

SQL allows you to filter your summarized results via the HAVING clause. The HAVING clause works in the same fashion as a SQL WHERE clause except that, instead of restricting which rows are processed, it restricts which post-summarized data are returned. In the example below, the SASHELP.CLASS table is summarized by SEX and AGE, producing a single record for each age within each sex type that has an average height greater than 60. The resulting records will be written to the table WORK.HEIGHT_SUM. Note that the HAVING clause evaluates a summarized value, avg(height), against a constant, 60. The summary function, which returns an average value for height, does so for each combination of values listed in the GROUP BY statement (i.e. SEX and AGE). If, for instance, the average height for girls who are 13 years old is greater than 60, then that record’s summarized height values as listed in the SELECT statement are written to WORK.HEIGHT_SUM.

select sex,
    age,
    count(*) as count,
    avg(height) as avg_height,
    min(height) as min_height,
    max(height) as max_height,
    std(height) as std_height
from sashelp.class
group by sex,
    age
having avg(height) > 60;

Combining Data:
There are many ways to combine data from two or more data sources. This paper will demonstrate how to control how data sources are combined using various SQL joins. There are two principle types of joins: inner and outer (full, right and left). An inner join produces a result set that contain records from all source tables that share common key column(s) values. The syntax to perform an inner join is listed below.

select a.index,
    a.field1,
    b.field2
from one a
inner join
    two b
on a.index=b.index;

Notice that between the source table names ONE and TWO, the join type, INNER JOIN, is listed. The join criterion, which defines which key columns proc SQL will compare, is listed in the ON statement. Also note that the letters 'a' and 'b', that immediately follow the table names in the FROM statement, are used as shorthand table references to distinguish between commonly named fields that exist on more than one table.
The results set will contain all records from WORK.ONE and WORK.TWO that have matching key column values stored in the fields INDEX. When proc SQL finds a match between INDEX from WORK.ONE and INDEX from WORK.TWO, FIELD1 from WORK.ONE is combined on the same result set record with FIELD2 from WORK.TWO.

An outer join writes as output, in addition to those records that match on key columns, records that do not have key column(s) matches. A full outer join returns all matching records, plus all records from all source tables that do not have key column(s) matches. A left outer join, will return all records from the table on the left hand side of the join type definition, and only those matching records from the table immediately to the right of the join type definition. A right outer join returns matching records from the table on the left hand side of the join type definition, and all records from the table immediately to the right of the join type definition.

In the sample SQL code below, we perform a left join between WORK.MASTER_LIST and WORK.DEATH_LIST on common values of social security number (SSN). SSN, LAST_PAY and LAST_PAY_DT for all rows of WORK.MASTER_LIST and DEATH_DT from any matching records on WORK.DEATH_LIST are returned.

```sql
select a.ssn,
       a.last_pay,
       a.last_pay_dt,
       b.death_dt
from master_list a
  left join
death_list b
on a.ssn = b.ssn;
```

Creating/Dropping Indexes:

An index is an object associated with a specific table that stores information on where in the said table the values of the indexed column or columns are stored. An index acts as a virtual sort, without affecting the physical order of data within its associated table. An index can greatly improve query performance, however its efficacy can vary due to a number of factors. Improving query performance is a complex subject matter, and is beyond the scope of this paper. I encourage the user to explore further how to improve query performance using indexes.

You can create multiple simple (single column) and composite indexes (two or more columns) for a given table. The syntax to create an index is listed below.

```
CREATE [UNIQUE] INDEX indexname ON
        table-name (column <,column> ...);
```

The optional UNIQUE keyword applies a constraint on the table, preventing any indexed columns from having duplicate values. You can not create a unique index on a column or columns that contain duplicate values.

The code below creates a simple index on the column SEX, and a unique composite index SEX_NAME on the columns SEX and NAME on the table SASHELP.CLASS.

```sql
create index sex
  on sashelp.class;
create unique index sex_name
  on sashelp.class(sex, name);
```

To drop an index, use the following SQL syntax.

```sql
DROP INDEX indexname <,indexname> ...
FROM table-name;
```

The code below drops the composite index SEX_NAME from SASHELP.CLASS.

```sql
drop index sex_name
  from sashelp.class;
```

Conclusions:

Proc SQL provides a SQL gateway to the SAS system. A general knowledge of proc SQL syntax allows the SAS programmer to exploit the variety of programming, querying and data management capabilities afforded by SQL. Also, this general knowledge can be used to reduce the learning curve for programming tasks in other non-SAS, SQL-based databases such as Oracle and DB2.

Contact Information:

Please feel free to send comments or questions to:

Christopher Zogby
Zogby Enterprises, Inc.
3901 Terrett Court
Alexandria, VA 22311

Phone: (703) 671.6330
Fax: (703) 671.6360
Email: info@zogbyenterprises.com
Web: www.zogbyenterprises.com

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.