Original Research Article
Journal der Pharmazie Forschung (**Formerly-Recent Advances in Pharmaceutical Science Research**)
Vol-2 | No-1 | 2013

Development of efficient SAS programs, table templates and data listings for preparing, processing and analyzing clinical data by using SQL

Vasanth Kumar Kunithala[*], Shiva Manoj kumar, Ravali Bojja, Mahendranath Ranga, Umasree Bharam, Puvvula Prathima and Ahamed Kabeer

International drug discovery and clinical research Private Limited, Road no.12, Banjarahills, Hyderabad-500034, Andhra Pradesh, India.

Abstract

The article is the overview of the Structured Query Language procedure on specifying columns, rows, data presenting, summarizing and subqueries. Mainly discussed about simple joins, complex joins and set operators for creating and modifying tables, views and indexes. It covers about how to maintain the SQL tables and dictionary tables and also setting the SQL procedure options and views about program testing and performance. The main advantage of the SQL tool is it is an augmentation to the DATA step. Comparison between PROC SQL Join versus DATA Step Merge. To check the PROC SQL view which is a stored PROC SQL query that retrieves the data that is stored in other files. Finally it describe about how to control and use the SQL tool to develop the efficiency SAS programs, table templates and data listings for preparing, processing and analyzing clinical data by using SQL .

Keywords: Structured query language, table templates, data listings, data validation, analyzing clinical data, queries report

*Corresponding author
vasanthmph@gmail.com

ACRONYM	DEFINITION
SQL	Structured Query Language
ANSI	American National Standards Institute
DBMS	Database Management System
DIF	Data Interchange Format

Structured Query Language (SQL) is a standardized language that is widely used to retrieve and update data in tables and in views based on those tables. It was originally designed as a query tool for relational databases, but it is now used by many software products. In 1970 it was conceptualized and proposed by Dr. E. F. Codd at the IBM Research Laboratory. In 1981 the first commercial SQL-based product, the IBM SQL/DS System was released. Finally in 1989 over 75 SQL database management systems exists, including SAS Release 6.06 version. The SQL procedure uses SQL to query SAS data sets, generate reports from SAS data sets, combine SAS data sets in many ways, create and delete SAS data files, views, indexes and update existing SAS data sets. The SQL procedure enables us to use SQL within the SAS and follows the guidelines set by the American National Standards Institute (ANSI) to include enhancements for compatibility with SAS software. SQL is a part of base SAS software and it can replace the need for multiple DATA and PROC steps with one query.SQL is not a replacement for the DATA step and it is not a custom reporting tool. It **is** a tool for queries, data manipulation and an augmentation to the DATA step. A SAS data set can be a SAS data file that stores data descriptions and data values together. PROC SQL view stores a PROC SQL query that retrieves the data stored in other files. DATA step view stores a DATA step that retrieves the data stored in other files. SAS/ACCESS view stores information required to retrieve data stored in a DBMS.
Some of PROC SQL options are :- BUFFERSIZE, CHECK, CODEGEN, CONSTDATETIME, DOUBLE, DQUOTE, ERRORSTOP, EXEC, EXITCODE, FEEDBACK, FLOW, INOBS,IPASSTHRU, LOOPS, NOCHECK, NOCODEGEN, NOCONSTDATETIME, NODOUBLE, NOERRORSTOP, NOEXEC,NOFEEDBACK, NOFLOW, NOIPASSTHRU, NONUMBER, NOPRINT, NOPROMPT, NOREMERGE, NOSORTMSG,NOSTIMER, NOTHREADS, NUMBER, OUTOBS, PRINT, PROMPT, REDUCEPUT, REDUCEPUTOBS, REDUCEPUTVALUES, REMERGE, SORTMSG, SORTSEQ, STIMER, THREADS, UNDO_POLICY.

BASIC QUERIES

A) OVERVIEW OF THE SQL PROCEDURE

1) Features of PROC SQL:-1) The PROC SQL statement does not need to be repeated with each query. 2) Each statement is processed individually 3) No PROC PRINT step is needed to view query results. 4) No PROC SORT step is needed to order query results.5) No RUN statement is needed.6) Use a QUIT statement to terminate PROC SQL.

2) Features of the SELECT Statement: - A SELECT statement is used to query one or more SAS data sets. 1) Selects data that meets certain conditions and groups data 2) Specifies an order for the data3) formats the data & queries from 1 to 256 tables.

Table names and variable names can be 1 to 32 characters in length and are not case-sensitive. But librefs, filerefs, formats and informats are limited to 8 characters.

3) Features of the VALIDATE Keyword:- The features of the VALIDATE keyword include the following:-1) It is used only in a SELECT statement 2) Tests the syntax of a query without executing the query 3) Checks column name validity 4) Prints error messages for invalid queries. The NOEXEC option checks for invalid syntax in all statements in SQL procedure but VALIDATE option apply only to the SELECT statement.

B) RETRIEVING DATA FROM A TABLE

1) Specifying Columns:- FEEDBACK option is used to write the expanded SELECT statement to the SAS log. After calculating new columns from existing columns, and name the new columns using the AS keyword, the new column is called an alias and the AS keyword is required. Omission of the alias causes the column leading to be blank. Use SAS DATA step functions for calculating columns. All SAS DATA step functions are supported except LAG, DIF.

2) Specifying Rows: - Use of the DISTINCT keyword to eliminate duplicate rows in query results. The DISTINCT keyword applies to all columns in the SELECT list. One row is displayed for each existing combination of values.

Subsetting with the WHERE clause:

WHERE is used as clause to specify a condition that the data must satisfy before being selected. We can use the IN operator to compare a value to a list of values. If the value matches at least one in the list, the expression is true; otherwise, the expression is false. Use of either CONTAINS or ? to select rows that include the substring specified. Use of either IS NULL or IS MISSING to select rows with missing values. With the = operator, we must know the column is character or numeric. On the other hand, if we use MISSING=, we do not need advance knowledge of column type. Use of BETWEEN-AND to select rows containing ranges of values. Use of LIKE to select rows by comparing character values to specified patterns [1-5]. A % sign replaces any number of characters. A single underscore ('_') replaces individual characters. The ESCAPE clause in the LIKE condition enables us to designate a single character string literal, known as an escape character, to indicate how PROC SQL should interpret the LIKE wildcards, percent (%) and underscore (_) if they are used within a character string. The sounds-like (**=***) operator selects rows containing a spelling variation of the specified word(s). Because a WHERE clause is evaluated first, columns used in the WHERE clause must exist in the table or being derived from existing columns. The only solution is to repeat the calculation in the WHERE clause. A more efficient method is to use the CALCULATED keyword to refer to already calculated columns in the SELECT clause. We can also use the CALCULATED keyword in other parts of a query, for example, in a SELECT clause. The Basic queries examples are shown in Table-1.

3) Presenting Data:-Use of the ORDER BY clause to sort query results in ascending order (default) or descending order by following the column name with the DESC keyword. We can specify the collating sequence by using the SORTSEQ=option in the PROC SQL statement. We use this option only when a collating sequence other than the systems or installations default collating sequence. In an ORDER BY clause, we order query results by specifying the following:1) any column or expression (display or non display) 2) a column name or a number that represents the position of an item in the SELECT list 3) multiple columns. We can use SAS formats and labels to customize PROC SQL output. After the column name in the SELECT list, we specify the following: LABEL= option to alter the column heading FORMAT= option to alter the appearance of the values in that column. The LABEL, FORMAT, INFORMAT and LENGTH options are not part of the ANSI standard but are SAS enhancements. To force PROC SQL to ignore permanent labels in a table, specify the NOLABEL system option. TITLE and FOOTNOTE statements must precede the SELECT statement [6].

Enhancing query output: 1) Define a column containing a character constant by placing a text string in the SELECT list 2) Use SAS titles and footnotes to enhance the query~~'s~~ appearance.

4) Summary functions: - If we specify only one column name in a summary function, the statistic is calculated down the column or if we specify more than one column name in a summary function the calculation is performed across columns for each row. The following are selected functions:1) AVG, MEAN shows the mean or average value 2) COUNT, FREQ shows the N number of non-missing values 3) MAX shows the largest value 4) MIN shows the smallest value 5)NMISS shows the number of missing values 6) STD shows the standard deviation 7) SUM shows the sum of values 8)VAR shows the variance. By default summary functions calculate statistics based on the entire table. The average is calculated and then re-merged with the individual rows in the table. Presenting data and summarizing data examples are shown in Table-2.

Table-1: Basic queries examples

The SELECT Statement

```
1)proc sql;
select EmpID, JobCode, Salary
from airline.payrollmaster
where JobCode contains 'NA'
order by Salary desc;
```

The VALIDATE Keyword

```
1)proc sql;
Validate
 select EmpID, JobCode, Salary
 from airline.payrollmaster
 where JobCode contains 'NA'
order by Salary desc;
NOTE: PROC SQL statement has valid syntax.
```

The NOEXEC Option

```
1)proc sql noexec;
select EmpID,JobCode,Salary
from airline.payrollmaster
where JobCode contains 'NA'
order by Salary desc;
NOTE:Statement not executed due to NOEXEC option.
```

Retrieving Data from a Table

```
1)proc sql;
select EmpID, JobCode, Salary
from airline.payrollmaster;
2)proc sql;
select *from airline.payrollmaster;
```

The FEEDBACK Option

```
1)proc sql feedback;
   select *
      from airline.payrollmaster;
NOTE: Statement transforms to
      select PAYROLLMASTER.EmpID,
             PAYROLLMASTER.Gender,
PAYROLLMASTER.JobCode,
             PAYROLLMASTER.Salary,
             PAYROLLMASTER.DateOfBirth,
             PAYROLLMASTER.DateOfHire
         from AIRLINE.PAYROLLMASTER;
```

Expressions

```
1)Proc sql;
select EmpID, JobCode, Salary,
Salary * .10 as Bonus.int((today()-
DateOfBirth)/365.25) as Age
from airline.payrollmaster;
```

Eliminating Duplicate Rows

```
1)proc sql;
SelectdistinctFlightNumber,
 Destination from airline.internationalflights;
```

Subsetting with the WHERE Clause

```
1)proc sql;
select EmpID,JobCode,Salary
from airline.payrollmaster
where Salary > 112000;
where JobCategory in ('PT','NA','FA')
where DayOfWeek in (2,4,6)
where word ? 'LAM'
where FlightNumber is missing
where FlightNumber is null
where FlightNumber = ' '
where FlightNumber =  .
where Date between '01mar2000'd
      and '07mar2000'd
where Salary between 70000 and 80000
where LastName =* 'SMITH'
where LastName like 'H%'
where JobCode like '__1'
```

ESCAPE Clause

```
1)proc sql;
select EmpID, Jobcode
from airline.payrollmaster2
where jobcode like 'FA/_%' ESCAPE '/';
quit;
```

Subsetting with Calculated Values

```
1)proc sql;
select FlightNumber,Date,Destination,
Boarded + Transferred + Nonrevenue
as Total from airline.marchflights
where Total < 100;
ERROR: The following columns were not found in the
contributing tables: Total.
2)proc sql;
select FlightNumber, Date, Destination,
Boarded+Transferred+Nonrevenue
as Total from airline.marchflights
where Boarded+Transferred+Nonrevenue < 100;
quit;
```

a) Grouping data:- We use the GROUP BY clause to 1) Classify the data into groups based on the values of one or more columns 2) Calculate statistics for each unique value of the grouping columns. The COUNT (*) summary function counts the number of rows. The COUNT function is the one that allows an asterisk (*) as an argument. The WHERE clause selects data based on values for individual rows. So to select entire groups of data, use the HAVING clause [7,8].

Table-2: Presenting data and summarizing data examples

Ordering Data
1)proc sql;
select FlightNumber, Date,
Origin, Destination,
Boarded+Transferred+Nonrevenue
from airline.marchflights
where Destination='LHR'
order by Date,5 **desc**;
2)proc sql;
select EmpID label='Employee Identifier', JobCode
label='Job Code',
Salary label='Annual Salary'
format=dollar12.2
from airline.payrollmaster
where JobCode contains 'NA'
order by Salary **desc**;
Enhancing Query Output
2)proc sql;
title 'Current Bonus Information';
title2 'Navigators - All Levels';
select EmpID
label='Employee Identifier',
'bonus is:',
Salary *.05 format=dollar12.2
from airline.payrollmaster
where JobCode contains 'NA'
order by Salary desc;
Summary Functions & Grouping Data
1)proc sql;
select avg(Salary) as MeanSalary
from airline.payrollmaster;
	2)proc sql;
	select JobCode, avg(Salary) as
	average format=dollar11.2
	from airline.payrollmaster
	group by JobCode;
	Analyzing Groups of Data
	1)proc sql;
	select **count(*)** as count
	from airline.payrollmaster;
	2)proc sql;
	select substr(JobCode,1,2)
	label='Job Category',
	count(*) as count
	from airline.payrollmaster group by 1;
	2)proc sql;
	select EmpID, Salary,
	(Salary/sum(Salary)) as percent format=percent8.2 from
	airline.payrollmaster
	where JobCode contains 'NA';
	Analyzing Groups of Data
	1)proc sql;
	select JobCode, avg(Salary) as average
	format=dollar11.2
	from airline.payrollmaster
	group by JobCode
	having avg(Salary) > 56000 ;
	2)proc sql;
	select JobCode,avg(Salary) as MeanSalary
	from airline.payrollmaster
	group by JobCode
	having avg(Salary) >(select avg(Salary)
	from airline.payrollmaster);

b) Subqueries: - Subqueries are also known as nested queries, inner queries and sub-selects. It has the following characteristics:-1) Inner queries that return values to be used by an outer query to complete a sub setting expression in a WHERE or HAVING clause 2) It return single or multiple values to be used by the outer query 3) It can return only a single column

i) Non-Correlated subqueries: - If we specify the ANY keyword before a subquery, the comparison is true. If it is true for any of the values that the subquery returns, that means maximum value in the subquery 5) The ALL keyword is true only if the comparison is true for all values returned, that means minimum value in the subquery.

ii) Correlated subqueries: - Correlated subqueries cannot be evaluated independently, but depend on the values returned by the outer query for their results and are evaluated for each row in the outer query**.** When a column appears in more than one table we must qualify each column with a table name or use alias to avoid ambiguity. The subqueries examples are shown in Table-

3. The EXISTS condition tests for the existence of a set of values returned by the sub query. The EXISTS condition is true if the sub query returns at least one row. The NOT EXISTS condition is true if the sub query returns no data.

Subqueries examples

Subqueries: Noncorrelated

```
1)proc sql;
select EmpID, LastName, FirstName,
City, State
from airline.staffmaster
where EmpID in(select EmpID
from airline.payrollmaster where
month(DateOfBirth)=2);
```

Selecting Data

```
1)proc sql;
title "FA1's or FA2's Older Than ANY FA3's";
select EmpID, JobCode, DateOfBirth
from airline.payrollmaster
where JobCode in ('FA1','FA2')
and DateOfBirth < any
(select DateOfBirth from airline.payrollmaster
where JobCode='FA3');
```

Correlated Subqueries

```
1)proc sql;
select LastName, FirstName, State
from airline.staffmaster where 'NA'=(select
JobCategory from airline.supervisors where
staffmaster.EmpID=supervisors.EmpID) ;
2)proc sql;
select LastName, FirstName
 from work.fa  where not exists( select *from
airline.flightschedule where fa.EmpID=
 flightschedule.EmpID) order by EmpID;
```

Cartesian Product

```
select*from one, two;
```

Inner Joins

```
3)select * from one, two
where one.X=two.X;
```

Outer Joins

```
4)select *
   from one left join two
   on one.X=two.X;
2)select one.X, a, b  from one, two
where one.X=two.X;
3)select* from one inner join two
On one.x=two.x;
4)title 'New York Employees';
 select substr(FirstName,1,1)||'. ' ||
 LastName as Name, JobCode,
 int((today()-DateOfBirth)/365.25)
 as Age from airline.payrollmaster,
 airline.staffmaster where payrollmaster.
 EmpID=staffmaster.EmpID
 and State='NY' order by JobCode;
5)select *
   from one right join two
   on one.X=two.X;
6)select *
   from one full join two
   on one.X=two.X;
title 'All March Flights';
7)proc sql;
select marchflights.Date, marchflights.FlightNumber
label='Flight Number', marchflights.Destination
label='Left', flightdelays.Destination  label='Right',Delay
from airline.marchflights
left join
airline.flightdelays
on marchflights.Date=flightdelays.Date
and marchflights.FlightNumber=flightdelays.FlightNumber
order by Delay;
```

Joined on inequalities

```
1)proc sql;
Select coiumns from tablel as a, Table-2 as b
Where a.itemnumber=b.itemnumber and a.cost > b.price;
```

COMBINING TABLES

Types of Joins:-

PROC SQL supports the following two types of joins:-

1) Inner joins: An Inner joins is sometimes called a conventional join, natural join or equijoin. Inner joins return only matching rows, it allow a maximum of 256 tables to be joined at the same time. If the join involves views, the number of tables underlying the views, not the views themselves, counts towards the limit of 256. Inner join syntax resembles, Cartesian product syntax, but it has a WHERE clause that restricts how the rows can be combined. Conceptually, PROC SQL performs the following tasks in inner joins. First builds a Cartesian product then applies the specified restriction(s) and removes rows. Inner joins display all combinations of rows with matching keys, including duplicates. For inner joins tables do not have to be sorted before they are joined.

Cartesian product: - A query that lists multiple tables in the FROM clause, without row restrictions, results in all possible combinations of rows from all tables.

Table-3: Creating and modifying tables and views

Using a Table Alias

```
2)select l.Date,
l.FlightNumber  label='Flight Number', l.Destination
label='Left',r.Destination label='Right',Delay
from airline.marchflights as l
 left join airline.flightdelays as r  on l.Date=r.Date and
 l.FlightNumber=r.FlightNumber order by Delay;
```

SQL Join versus DATA Step Merge

```
1)data merged;
merge one two; by X;
run;
2)proc sql;
select one.X, a, b
from one full join two
on one.X=two.X;
3)select coalesce(one.X,two.X)
 label='X', a, b
from one full join two
on one.X=two.X;
```

In-Line Views

```
1)select *, Late/(Late+Early) as prob
 format=5.2 label='Probability of Delay'
 from (select Destination,
 avg(Delay) as average
 format=3.0 label='Average Delay',
 max(Delay) as max
 format=3.0 label='Maximum Delay',
 sum(Delay > 0) as late
 format=3.0 label='Number of Delays',
 sum(Delay <= 0) as early
format=3.0
 label='Number of Early Arrivals'
 from airline.flightdelays
 group by 1)
 order by 2;
```

Handling a Complex Query

```
2)select FirstName, LastName
 from airline.staffmaster where EmpID in
 {select EmpID from airline.supervisors as m,
[select substr(JobCode,1,2) as JobCategory, State
 from airline.staffmaster as s, airline.payrollmaster as p
 where s.EmpID=p.EmpID and s.EmpID in (select
EmpID
 from airline.flightschedule where
Date='04mar2000'd and
 Destination='CPH')] as c where
m.JobCategory=c.JobCategory and m.State=c.State};
2)select distinct e.FirstName, e.LastName
 from airline.flightschedule as a,
 airline.staffmaster as b,
 airline.payrollmaster as c,
 airline.supervisors as d,
 airline.staffmaster as e
 where a.Date='04mar2000'd and
 a.Destination='CPH' and
 a.EmpID=b.EmpID and
 a.EmpID=c.EmpID and
d.JobCategory=substr(c.JobCode,1,2)
 and d.State=b.State and
 d.EmpID=e.EmpID;
```

This is called a Cartesian product. The number of rows in a Cartesian product is the product of the number of rows in the contributing tables. A Cartesian product is rarely a desired query outcome. The SQL processor prints a warning in the log if a query involved a Cartesian product. Creating and modifying tables and views examples was shown in Table-4.

2) Outer joins: - An outer join is an augmentation of an inner join. It returns all the rows generated by an inner join, plus others. Outer joins return all matching rows, as well as non-matching rows from one or both tables; it can be performed on only two tables or views at a time. The distinguishing characteristics of outer join syntax are exactly two table names flanking one of the three JOIN operators in the FROM clause and a special ON clause specifying the join condition. The AS keyword is a table alias, the alias can directly follow the table name in the FROM clause. In the SQL procedure, the two X columns are overlaid by default, but we can achieve them by using COALESCE function. It returns the first value that is a SAS non-missing value and requires all arguments to have the same data type. if we omit the LABEL= option or an alias in a coalesced column, it appears without a column heading. Tables can be joined on inequalities. The SQL procedure optimizer can process an equijoin more efficiently than a join involving an inequality.

3) Complex Joins: - Includes techniques that simplify the coding of a complex query and inline views. An inline view has the following characteristics:- 1) A temporary table that exists only during query execution 2) Created when a FROM clause contains a query expression in place of a table name. Boolean expressions can be used in the SELECT clause. Boolean expressions resolves either to 1(true) or 0(false).

Table-4: Joins the tables using set operators and modifiers examples

Types of Set Operators & Modifiers
1)select * from one **except** select * from two;
2)select * from one
except all select *from two;
3)select *from one **except corr**
select * from two;
4)select FirstName, LastName
from airline.staffchanges **except all**
select FirstName, LastName
from airline.staffmaster;
5)select count(*) label='No. of Persons'
from (select EmpID from airline.staffmaster
except all select EmpID from airline.staffchanges);
6)select * from on **intersect**
select * from two;
7)select * from one **intersect corr**
select * from two;
8)select FirstName, LastName
from airline.staffmaster **intersect all**
select FirstName, LastName
from airline.staffchanges;
9)select *
from one **union** select * from two;
10)select *from one **union corr**
select *from two;
11)title 'Points and Miles Traveled ' 'by Frequent Flyers';
select 'Total Points Earned :',sum(PointsEarned)
format=comma12.from airline.frequentflyers
	union all
	select 'Total Points Used :', sum(PointsUsed
	format=comma12. from airline.frequentflyers
	union all
	select 'Total Miles Traveled:',sum(MilesTraveled
	format=comma12. from airline.frequentflyers;
	12)select *from one **outer union**
	select * from two;
	13)select * from one **outer union corr**
	select * from two;
	14)select * from airline.mechanicslevel1
	outer union corr select * from airline.mechanicslevel2
	outer union corr select *from airline.mechanicslevel3;
	SQL versus Traditional SAS Programming
	1)data allmechanics;
	set mechanicslevel1 mechanicslevel2;
	run;
	proc print data=allmechanics noobs;
	run;
	2)proc sql;
	select * from mechanicslevel1
	outer union corr
	select * from mechanicslevel2;
	quit;
	3)proc append base=mechanicslevel1
	data=mechanicslevel2;
	run;
	proc print data=one noobs;
	run;

a)Handling a Complex Query:-

An example for handling a complex query what are the names of the supervisors for the crew on the flight to Copenhagen on March 4, 2000?. Step 1:- Identify the crew for the flight. Step 2:- Find the states and job categories of the crew returned from the first query. Step 3:-Find the employee numbers of the crew supervisors based on the states and job categories generated by the second query. Step 4:-Find the names of the supervisors based on the employee numbers returned from the third query.

b) Choosing Between SQL Joins and DATA Step Merges:-

1) DATA step merges are usually more efficient than SQL joins in combining small tables. 2) SQL joins are usually more efficient than DATA step merges in combining large, unsorted tables. 3) SQL joins are usually more efficient than DATA step merges in combining a large, indexed table with a small table.4) For ad hoc queries, select the method that we can code in the

shortest time. 5) For production jobs, experiment with different coding techniques and evaluate performance statistics[3].

Table-5: Loading Data into a Table examples

Creating Tables

```
1)proc sql;
create table airline.discount (destination char(3),Begin
Date num format=date9.,EndDate num
format=date9.,Discount num);
2)Proc sql;
create table airline. delaycat
(drop=DelayCategory DestinationType)
      like airline.flightdelays;
3)proc sql;
create table airline.fa as
select LastName, FirstName, Salary
from airline.payrollmaster, airline.staffmaster
where payrollmaster.EmpID =staffmaster.EmpID and
JobCode contains 'FA' ; select *from airline.fa;
```

Loading Data into a Table

```
1)proc sql;
 insert into discount
 set Destination='LHR', BeginDate='01MAR2000'd,
EndDate='05MAR2000'd,Discount=.33
 set Destination='CPH',
BeginDate='03MAR2000'd,
EndDate='10MAR2000'd, Discount=.15;
2)proc sql;
insert into discount
values('LHR','01MAR2000'd,
            '05MAR2000'd,.33)
values('CPH','03MAR2000'd,
            '10MAR2000'd,.15);
3)proc sql;
insert intodiscount (Destination,Discount)
 select Destination, Rate*.25 from airline.fares where
Type='special';
4)proc sql;
 create table discount (Destination char(3),BeginDate
date label='BEGINS',
EndDate date label='ENDS',Discount num);
insert into discount
values('LHR','01MAR2000'd,'05MAR2000'd,.33)
values('CPH','03MAR2000'd,'10MAR2000'd,.15)
values('CDG','03MAR2000'd,'10MAR2000'd,.15)
values('LHR','10MAR2000'd,'12MAR2000'd,.05);
airline.payrollmaster,
airline.staffmaster where JobCode contains 'FA' and
staffmaster.EmpID=
payrollmaster.EmpID;
```

Create Integrity Constraints

```
proc sql;
create table discount(Destination char(3), Begin Date
date label='BEGINS', End Date date label='ENDS',
Discount num,
      CONSTRAINT ok_discount check
         (Discount le .5));
```

Rollbacks with the UNDO_POLICY Option

```
proc sql undo_policy=none ;
 insert into discount
values('CDG','03MAR2000'd,'10MAR2000'd,.15)
values('LHR','10MAR2000'd,'12MAR2000'd,.55);
```

Documenting Table Definitions and Integrity Constraints

```
proc sql; describe table discount;
```

Creating a View

```
1) proc sql;
create view airline.faview as
select LastName, FirstName, Gender, int((today()-
DateOfBirth)/365.25) as Age,substr(JobCode,3,1) as
Level, Salary from
2)proc sql;
select *from airline.faview;
3) proc tabulate data=airline.faview;
class Level;
var Age;
table Level*Age*mean;
run;
4)proc sql;
describe view airline.faview;
5)proc sql;
create view sasdata.master as
 select *from sasdata.payrollmaster;
```

Administering Views: Using the Embedded LIBNAME Statement

```
libname sasdata 'SAS-data-library-one';
libname airline 'SAS-data-library-two';
1)proc sql;
create view sasdata.journeymen as
select *from airline.payrollmaster
where JobCode like '__2'
using libname airline 'SAS-data-library-three';quit;
proc print data = sasdata.journeymen ;
run;
2)create view manager.info as
select *from fa1.info
outer union corr select *from fa2.info
outer union corr select *from fa3.info;
```

4) Set Operators:-

Set operators combine rows from two tables vertically. The following are the four set operators:- 1) EXCEPT: - Unique rows from the first table that are not found in the second table are selected. 2) INTERSECT: - Common unique rows from both tables are selected. 3) UNION: - All unique rows from both tables are selected with columns overlaid. 4) OUTER UNION: - All rows from both tables, unique as well as non-unique, are selected and columns are not overlaid. In EXCEPT, INTERSECT, UNION columns are matched by position and must be the same data type. Column names in the result set are determined by the first table. Joining~~s~~ the tables using set operators and modifiers examples are shown in Table-5.

Table-6: Maintaining tables examples

Creating an Index

```
1) proc sql;
create unique index EmpID
on airline.payrollmaster(EmpID);
2)proc sql;
create unique index daily
on airline.marchflights(FlightNumber,Date);
3) options msglevel = i;
proc sql;
select *from airline.payrollmaster
where JobCode = 'NA1';
INFO:Index JobCode selected for WHERE clause
optimization.
select *
      from airline.payrollmaster
      where Salary gt 100000;
4)proc sql;
   drop index EmpID
        from airline.payrollmaster;
NOTE: Index EmpID has been dropped.
   drop table Discount;
NOTE: Table WORK.DISCOUNT has been
dropped.
```

Updating Data Values

```
1) update one set x=x*2 where y contains 'a';
2)proc sql;
update airline.payrollmaster
 set Salary=Salary * 1.05 where JobCode like
'__1';s elect *rom airline.payrollmaster;
3) proc sql;
update airline.payrollmaster
set Salary=Salary *
case substr(JobCode,3,1)
when '1' then 1.05
when '2' then 1.10
when '3' then 1.15
 else 1.08   end;
4)proc sql;
update airline.payrollmaster
set Salary=Salary *
case when substr(JobCode,3,1)='1' then 1.05
when substr(JobCode,3,1)='2' then 1.10
when substr(JobCode,3,1)='3' then 1.15
else 1.08 end;
5)proc sql;
select LastName, FirstName, JobCode,
case substr(JobCode,3,1)
when '1' then 'junior'
when '2' then 'intermediate'
when '3' then 'senior'
else 'none' end as level
from airline.payrollmaster,
airline.staffmaster where staffmaster.EmpID=
 payrollmaster.EmpID;
```

Deleting Rows

```
1) proc sql;
delete from airline.frequentflyers
where PointsEarned-PointsUsed <= 0;
NOTE: 11 rows were deleted from
AIRLINE.FREQUENTFLYERS.
2) delete from one where y contains '1';
```

Modifiers: - We can use the following two keywords to modify the behavior of set operators:1) ALL 2) CORRESPONDING. The following are characteristics of the ALL keyword: It does not remove duplicate rows, and so avoids an extra pass through the data. Use the ALL keyword for better performance when it is possible. And this is not allowed in connection with an OUTER UNION operator. (It is implicit). The following are characteristics of the CORRESPONDING keyword: - overlays columns by name, instead of by position. It removes any columns not found in both tables when used in EXCEPT, INTERSECT, and UNION operations. And it causes common columns to be overlaid when used in OUTER UNION operations. It can be abbreviated as CORR.

Comparing Methods of Combining Tables Vertically:- 1) PROC APPEND is the fastest method of performing a simple concatenation of two tables. The BASE= table is not completely read; only the DATA= table is completely read. 2) When logical conditions are involved, we can choose either the DATA step or PROC SQL.3) SQL set operators generally require more computer resources, but the other operators are more convenient and flexible.4) With the DATA step, we can process an unlimited number of tables at one time. 5) With SQL set operators, we can work on only two tables at a time. 6) If multiple DATA steps are required to perform the task, consider using PROC SQL.

Table-7a: Setting PROC SQL options programs

```
Altering Columns
1)proc sql;
alter table airline.payrollmaster
add Bonus num format=comma10.2, Level char(3);
2)proc sql;
alter table airline.flightdelays
drop DestinationType;
3)proc sql;
alter table airline.payrollmaster
modify Bonus num format=comma8.2,
Level char(1) label='Employee Level';
3)proc sql;
alter table airline.payrollmaster
add Age num
modify DateOfBirth date format=mmddyy10.
drop DateOfHire;
update airline.payrollmaster
 set Age=int((today()-DateOfBirth)/365.25);
Controlling Processing
1)proc sql flow=13 double;
select * from awards;
2)proc sql inobs=10;
 select FlightNumber, Date
 from airline.marchflights;
```

```
3)proc sql outobs=2 number;
  select * from airline.payrollmaster;
4)reset nonumber;
select *
from airline.payrollmaster;
5)describe table dictionary.tables;
6)options nolabel nocenter;
5)proc sql;
select memname format=$20.,nobs,nvar,crdate
from dictionary.tables
where libname='AIRLINE';
select memname
from dictionary.columns
where libname='AIRLINE' and name='EmpID';
6)proc tabulate data=sashelp.vmember format=8.;
class libname memtype;
keylabel N=' ';
table libname, memtype/rts=10 misstext='None';
run;
7)%let datasetname=payrollmaster;
proc sql feedback noexec;
select *from airline.&datasetname;
```

CREATING AND MODIFYING TABLES AND VIEWS

1) Creating Tables: - We can use the CREATE TABLE statement in three ways. 1) Creates an empty table. There are two methods. A) Method 1A:- Define the columns and fill in the data rows later. B) Method 1B:- Copy a table. Use column definitions from another table and fill in the rows of data later. 2) Store a query result in a table that defines both columns and rows. Use method 1A and 1B to create tables containing columns that do not already exist in other tables, that is, define our own columns. Method 2 is particularly helpful when we create subsets or supersets of tables. Use of the CREATE TABLE statement shuts off the automatic report generation. Also, this is the only method of the three that does both creates and populates a table at the same time. Use this method to create a table which is similar or identical to another existing table.

Defining Columns: - PROC SQL accepts: - 1) Types of CHARACTER or VARCHAR, but interprets both as SAS CHARACTER. Default length is 8 bytes. 2) Types of INTEGER, SMALLINT, DECIMAL, NUMERIC, FLOAT, REAL, and DOUBLE PRECISION, interpreting all as SAS NUMERIC with a length of 8 bytes. 3) A type of DATE, interpreted as a SAS NUMERIC, with a length of 8 bytes and a DATE, Informat and format. Although SAS reads all of the above mention data types, only CHARACTER and NUMERIC are used in SAS tables.

Table-7b: General syntax of overall SQL procedures, statement and options.

```
0)PROC SQL statement =  dr-ci-suddar)
    DESCRIBE expression
    RESET <option <option
    CREATE expression;
    INSERT expression;
    SELECT expression;
    UPDATE expression;
    DROP expression;
    DELETE expression;
    ALTER expression;(ROLLBACK statement
USING statement)
1)PROC SQL;
   DESCRIBE TABLE table-name<,table-name>…;
   DESCRIBE VIEW proc-sql-view <, - view>…;
…DESCRIBE TABLE  CONSTRAINTS table-name
   DESCRIBE TABLE DICTIONARY.TABLES;
2) RESET <option <option
3)CREATE TABLE table-name (column-name
              type(length), <column-name type(length)>,... );
  CREATE TABLE table-name LIKE table-name;
  CREATE TABLE table-name AS query-expression;
  CREATE TABLE table  (column-specification,   .            .
<constraint specification,…>);
  CREATE VIEW view-name AS query-expression;
  CREATE VIEW proc-sql-view AS query-expression            .
USING statement<, libname-clause> . ;
  CREATE UNIQUE INDEX index-name
             ON table-name(column-name, column-name);
4)Method A: The SET Clause
   INSERT INTO table
               SET column-1=value,column-2=value,...;
   Method B: The VALUES Clause
   INSERT INTO table <(column-list)>
               VALUES (value,value,value, ...);
   Method C: A Query-expression
   INSERT INTO table-1 <(column-list)> 5a)SELECT columns
FROM table-2;
              FROM table-1|view-1<, table-2|view-2>...
              WHERE expression>
              GROUP BY column-1<, column-2>…>
              HAVING expression>
              ORDER BY column-1<, column-2> DESC;
```

```
5b)SELECT column <, column> …
             FROM table1
           INNER|LEFT|RIGHT|FULL JOIN
             table2
             ON join-condition(s)<other clauses>;
5c)SELECT column-1<, column-2> ...
           CASE <case-operand>
           WHEN when-condition THEN result-
           WHEN when-condition THEN result-
           ELSE result-expression>END as column
           FROM table;
           WHERE expression;
6)UPDATE table-name
           SET column-name=expression,
                  column-name=expression,…
           WHERE expression;
7)DROP TABLE table-1, table-2, …;
   DROP VIEW view-1, view-2, …;
   DROP INDEX index-1, index-2, FROM table;
8)DELETE FROM tanle-name WHERE expression;
9)ALTER TABLE table
           ADD column-definition, column-definition,
           DROP column-1, column-2, …
           MODIFY column-definition, column-
10)PROC SQL <option
           VALIDATE, NOEXEC, DISTINCT,
UNDO_POLICY=NONE, INOBS=1,  .      .            OUTOBS=2,
LOOPS=3, PROMPT=4,              NOPRINT,
NONUMBER, DOUBLE,               FLOW=14, STIMER;
11) options:-  MSGLEVEL=1 ;
12)keywords:-
           VALIDATE, UNIQUE, ALL, ANY
EXIST , NOT EXISTS, CORR,
CONSTRAINTS
13)set operators:-
           EXCEPT INTERSECT UNION
OUTER UNION
14)integrity constraints:-
a) general:-   1)NOT NULL 2)CHECK 3)UNIQUE
b)referential:-1)PRIMARY KEY 2)FOREIGH KEY
```

2) Loading Data into a Table

After the table is created, we can enter rows of data using the INSERT statement with one of the three methods. Method A: - The SET Clause. Method B: - The VALUES Clause. Method C: - A Query-expression. We may nest a SELECT statement within a SET statement in method A. Loading Data into Table examples was shown in Table-6.

a) Integrity Constraints: - Integrity constraints are rules that table modifications must follow to guarantee validity of data. We can preserve the consistency and correctness of data by specifying integrity constraints for a SAS data file. SAS uses the integrity constraints to validate data when we insert or update the values of a variable for which we have defined integrity constraints. Integrity constraints are part of Version 8 of Base SAS software and follows the ANSI standards. It cannot be defined for views and can be specified when a table is created or later when a table contains data. There are five Integrity Constraints A) General: 1) NOT NULL 2) CHECK 3) UNIQUE B) Referential: 1) PRIMARY KEY 2) FOREIGN KEY

b) Rollbacks: - If an INSERT or UPDATE statement experiences an error while it processes the statement, then the inserts or updates that were completed up to the point of the error by that statement can be undone by use of the UNDO_POLICY option. The ROLLBACK statement is although ANSI standard is not currently supported in the SQL procedure.

c) Rollbacks with the UNDO_POLICY Option:-1) UNDO_POLICY=REQUIRED (default) Undoes all inserts or updates that have been done to the point of the error. Sometimes the UNDO operation cannot be done reliably. 2) UNDO_POLICY=NONE prevents any updates or inserts from violating a constraint. 3) UNDO_POLICY=OPTIONAL reverses any updates or inserts that it can reverse reliably.

d) Documenting:-

The DESCRIBE statement displays the definition of the view or CREATE TABLE statement of a table. The DESCRIBE statement writes a CREATE TABLE statement to the SAS log for the specified table regardless of how the table was originally created. If table contains an index, CREATE INDEX statements for those indexes are also written to the SAS log. The DESCRIBE TABLE CONSTRAINTS statement writes table contained constraints information to the log.

3) Creating a View

A PROC SQL view: 1) It is a stored query. It contains no rows of data.2) It can be used in SAS programs in place of an actual SAS data file.3) It can be derived from one or more tables, PROC SQL views, DATA step views, or SAS/ACCESS views.4) It extracts underlying data when used, thus accessing the most current data. Views are not separate copies of the data and are referred to as virtual tables because they do not exist as independent entities as do real tables. It may be helpful to think of a view as a movable frame or window through which we can see the data. Thus when the view is referenced by a SAS procedure or in a DATA step, it is executed, and conceptually, an internal table is built. PROC SQL processes this internal table as if it were any other table.

Using a View in a 1) proc sql 2) proc tabulate 3) data step. we can1) Access the most current data in changing tables, DATA step views, or SAS/ACCESS views 2) Pull together data from multiple database tables or even different databases 3) Simplify complex query-expressions and prevent users from altering code and avoid storing a SAS copy of a large table.

Administering Views: - Some General Guidelines1) Avoid the ORDER BY clause in a view definition. Otherwise, the data must be sorted each time the view is referenced.2) If the same data is used many times in one program, create a table rather than a view. 3) Avoid specifying two-level names in the FROM clause when we create a permanent view that resides in the same library as the contributing table(s). For creating Views an alternative method: Embed the LIBNAME statement within a USING clause. This enables us to store a SAS libref in the view and does not conflict with an identically named libref in the SAS session

4) Creating Indexes: - The index is a structure that boosts program performance by serving as a logical pointer to a physical location of a given values. An index is an auxiliary data structure that specifies the location of rows based on the values of one or more key columns. The SQL procedure can utilize an available index to optimize sub setting or joining tasks. We have two types of indexes 1) Simple based on values of only one column. This simple index is based on EmpID and allows no duplicate ID numbers in the table. Names must match for a simple index 2) Composite based on values of more than one column concatenated to form a single value, for example, Date and Flight Number. The composite index named DAILY is based on Flight

Number and Date. A composite index cannot have the same name as a variable (column name). Controlling index usage in a WHERE Expression use two data set options: 1) IDXWHERE=YES (Forces index usage)| NO (Prevents index usage) 2) IDXNAME=<name>.Use of the optional UNIQUE keyword ensures that values in the row are unique. If a table contains multiple occurrences of the same value, the UNIQUE keyword is not accepted and the index is not defined on that column. Similarly, if we already have a uniquely defined index on a column and attempt to add a duplication value to the table, the row is not inserted. Indexes can be based on either a character or numeric variable, we do not want to create two indexes on the same variable. We can achieve improved index performance if we create the index on a pre-sorted data set. To determine if an index is used, specify the SAS system option MSGLEVEL=1.A note appears in the SAS log when an index is selected for processing.

Indexing and Performance: - Suggested guidelines for using indexes: 1) Keep the number of indexes to a minimum to reduce disk storage and update costs. 2) Do not create an index for small tables; sequential access is faster on small tables. 3) Do not create an index based on columns with a small number of distinct values, for example, Male and Female. 4) An index performs best when it retrieves a relatively small number of rows, that is, <=15%.

Benefits: - 1) Fast access to a small subset of data (<15%). 2) Equijoins can be performed without internal sorts. 3) Can enforce uniqueness. 4) BY group processing without sorting.

Costs :- 1) Extra CPU cycles and I/O operations to create an index.2) Extra disk space to store the index file.3)Extra memory to load index pages and code for use. 4) Extra CPU cycles and I/O operations to maintain the index.

5) Maintaining Tables: Overview

We can use PROC SQL to perform the following: 1) Modify values in a table or view 2) Add rows to a table or view 3) delete rows from a table or view 4) Alter column attributes of a table 5) add new columns to a table 6) Drop columns from a table 7) Delete an entire table, SQL view, or index. Use the UPDATE statement to modify column values in existing rows of a table or SAS/ACCESS view. We can also use a CASE expression in other parts of a query, such as within a SELECT statement, to create new columns. Use the DELETE statement to eliminate unwanted rows from a table or SAS/ACCESS view. Use the ALTER statement to manipulate columns in a table in three different ways. 1. Add columns to a table. 2. Drop columns from a table. 3. Modify attributes of existing columns in a table. We can alter a column’s length, informat, format and label. Use the DROP statement to delete an entire table, SQL view, or index. Maintaining tables examples is shown in Table-7a and 7b.

Updating Views:- We can update the data underlying PROC SQL views using the INSERT, DELETE, and UPDATE statements, but1) we can only update a single table through a view. It cannot be joined or linked to another table, nor contain a subquery.2) we can update a column using the column’s alias, but not a derived column.3) we cannot update the table through a summary query.4) we cannot update a view containing an ORDER BY clause.

ADDITIONAL SQL FEATURES

1) Setting SQL Procedure Options

The SQL procedure offers a variety of options and statements that affect processing. Selected options: 1) INOBS=n sets a limit of n rows from each source table that contributes to a query.2) OUTOBS=n restricts the number of rows that a query outputs (displays or writes to a table). 3) PRINT|NOPRINT controls whether the results of a SELECT statement are displayed. 4) NONUMBER|NUMBER controls whether the row number is printed as the first column in the output.5) NODOUBLE| DOUBLE double-spaces the report. 6) NOFLOW|FLOW| controls the appearance of FLOW=n| FLOW=n m wide character columns. The FLOW option causes text to be flowed in its column rather than wrapping the entire row. Specifying n determines the width of the flowed column. Specifying n and m floats the width of the column between the limits to achieve a balanced layout. We can use the RESET statement to add or change PROC SQL options without re-invoking the procedure. Setting PROC SQL options examples was shown in table8.

2) Dictionary Tables and Views

We can retrieve information about SAS session metadata by querying dictionary tables with PROC SQL. Dictionary tables follow these rules: 1) Created at initialization 2) Updated automatically 3) Limited to read-only access. The metadata available in dictionary tables includes the following: 1) SAS files2) external files3) system options, macros, titles, and footnotes[5]

Using of Dictionary Information:-To use session metadata in other procedures or in a DATA step, we can do the following:1) Create a PROC SQL view based on a dictionary table 2) Use views provided in the SASHELP library that are based on the dictionary tables

3) Program Testing and Performance

PROC SQL statement options are available to aid in testing programs and evaluating performance. The following are selected options: 1) EXEC|NOEXEC controls whether submitted SQL statements are executed.2) NOSTIMER|STIMER reports performance statistics in the SAS log for each SQL statement.3) NOERRORSTOP|ERRORSTOP is used in batch and no-interactive jobs to make PROC SQL enter syntax-check mode after an error occurs. Display the columns that are retrieved when we use SELECT * in a query and display any macro variable resolutions, but do not execute the query. This is a log from a PROC SQL step with the STIMER statement option that executes a single query. General syntax of overall SQL procedures, statement and options was shown in table9.

4) General Guidelines for Benchmarking Programs

1) Never use elapsed time for comparison because it might be affected by concurrent tasks. 2) Benchmark two programs in separate SAS sessions. If benchmarking is done within one SAS session, statistics for the second program can be misleading because the SAS supervisor might have loaded modules into memory from prior steps.3) Run each program multiple times and average the performance statistics.4) Use realistic data for tests. Program A could be better than program B on small tables and worse on large tables[9,10].

Conclusion

Finally we concluded that SQL is a modular language because queries or statements are composed of smaller building blocks or clauses. Using SQL tool is very convenient and time reducing tool for simple joins, complex joins and creating and modifying tables, views and Indexes.

Acknowledgement

We am very glad to take this opportunity of expressing my sincere thanks and gratitude to Mr. Herin and Sirisha Battu for their valuable guidance, constructive criticism, support and making the requisite arrangements complete successfully for our review article and there is no word to express our gratitude and thanks to our colleagues Karthik, Suresh, Anusha, Ravali, Umasree who have made my dream come true by continuous encouragement and back-up.

Reference

1. Prairie, Katherine. 2005. The Essential PROC SQL Handbook for SAS® Users. The Essential PROC SQL Handbook for SAS Users. Cary, NC: SAS Institute Inc.
2. Title, SAS SQL 1: Essentials: Course Notes. Authors, Davetta Dunlap, Mark Jordan. Contributor, SAS Institute. Publisher, SAS Institute, 2009. ISBN, 1607642425 .
3. https://support.sas.com/edu/schedules.
4. SAS SQL 1: Essentials, 4, Backlinks to support.sas.com, 6 . vrcgi.com, Programmers 'Refer, 43, 2012-06-05.
5. Delwich, Lora D. and Susan J. Slaughter. 2003. The Little SAS® Book: A Primer, Third Edition. Cary, NC: Sas Institute Inc.
6. https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&ved=0CEEQFjAD&url=http%3A%2F%2Fwww2.sas.com%2Fproceedings%2Fsugi26%2Fp129-26.pdf&ei=mcdbUd-JBsrRrQfT-CYAQ&usg= AFQjCNHLY_ vrdqY4B VoucRD Cm8czwd_Jag&bvm=bv.44697112,d.bmk
7. www2.sas.com/proceedings/sugi31/123-31.pdf
8. www2.sas.com/proceedings/sugi25/25/hands/25p146.pdf
9. www.hollandnumerics.co.uk/pdf/Efficient_SAS_Coding(paper)3.pdf
10. www.sconsig.com/sastips/tip00320.pdf