Chapter 1
Introduction

1.1 More about SQL 2
1.2 More about This Book 5

SAS defines Structured Query Language (SQL) as “a standardized, widely used language that retrieves data from and updates data in tables and the views that are based on those tables” (see *Base SAS 9.2 Procedures Guide*: Procedures: The SQL Procedure: Overview). SQL is not an exclusive feature of SAS; it has been implemented by many vendors, and is especially widespread in the relational database management system (RDBMS) world. The SAS implementation of SQL is available in the SQL procedure (PROC SQL), part of Base SAS.

Some but not all PROC SQL capabilities are paralleled in the DATA step and in other SAS procedures. Thus, PROC SQL can be employed as a substitute for other elements of SAS or as a complement to those elements.

This book is intended for readers who are familiar with SAS but not with SQL, and who want to add PROC SQL to their SAS toolkits. It will also be useful to those familiar with other implementations of SQL who want to learn SAS.

1.1 More about SQL

PROC SQL is different from other SAS components and different from other software implementations of SQL.

Standards and Extensions

American National Standards Institute (ANSI) standard SQL is not a complete and self-sufficient language. For example, consider the definition quoted in the preceding section; it mentions retrieval and updating of data held in tables, but says nothing about how a table is populated in the first place. There are two possible approaches to the incompleteness. One is to include extensions (capabilities not required by the standard) in an SQL implementation to make the language more complete. Thus, for example, RDBMS vendors typically extend SQL with tools to import and export large volumes of data. The other approach, and the one followed by SAS, is to embed SQL into a language that provides the missing features. So, for example, a SAS application might use a PROC IMPORT step to load data, before turning to PROC SQL for processing and analysis of that data.

The implementation of SQL in SAS 9.2 PROC SQL does not fully comply with the current ANSI standard for SQL. On the other hand, PROC SQL includes some features **not** required by the standard.

Reference: For details about PROC SQL and the ANSI standard, see *Base SAS 9.2 Procedures Guide*: Procedures: The SQL Procedure: PROC SQL and the ANSI Standard.

Fundamental Differences between SQL and the DATA Step

The largest part of this book is devoted to explaining and illustrating the features of PROC SQL and identifying and qualifying parallels with non-SQL SAS counterparts to those features. Those explanations and examples deal with particular language elements. Before delving into that sort of detail, we should look at some general characteristics that distinguish PROC SQL from other parts of SAS. These distinctions range from the rather mundane to the almost profound.

Comma versus White Space Separation

In most parts of SAS, a series of like elements (such as variable names) is coded using white space (blanks, tabs, or new lines) for separation. In SQL, elements in such a series are separated by commas (with optional white space permitted in addition to, but not instead of, each comma).

Terminology

SAS and SQL both use two-dimensional collections of data, but have different terminology for the basic elements. The differences are summarized in the following table, which, for reference, extends to include the realms of data processing and relational database theory.

Realm	Corresponding Terms		
SAS	SAS data file	Observation	Variable
SQL	Table	Row	Column
Data processing	File	Record	Field
Relational database theory	Relation	Tuple	Attribute

The SQL and SAS terms are used more or less interchangeably throughout the book. We usually use SQL terms in discussing SQL code and SAS terms in discussing other code, but that's just a tendency and not a rule.

The term "SAS data file," though taken from the SAS documentation, might seem unfamiliar; perhaps you more often refer to a "SAS data set." The distinction is the inclusion of views. A SAS data set might be either a SAS data file or a SAS data view. In SQL, there is no similar umbrella term; there are tables and there are views.

In the remainder of the book, you will encounter many references to SAS data sets, and few if any to SAS data files. Meanings should be clear in context. In places where it's important to differentiate views from files (particularly in Chapter 10, which has views as its subject), more precise language is used.

Reference: For more information about SAS data files, see *SAS 9.2 Language Reference: Concepts*: SAS Files Concepts.

Executable Unit

In the DATA step, and in most (but not all) SAS procedures, the step is the executable unit of code. That is, SAS reads and interprets code from the beginning of a step and continues until it encounters another step boundary before it begins processing. In SQL, each **statement** is an executable unit. So SAS reads and interprets code until it encounters a semicolon (which is the statement terminator for SQL just as it is elsewhere in SAS) and then performs the requested processing before examining the next statement. The QUIT statement terminates PROC SQL (RUN statements are ignored).

Nestability
SQL code constructs can be nested; that is, a query can be an operand of another query (see Section 3.3 and Chapter 5). In the DATA step and other parts of SAS, such sequencing of computations can be implemented only by coding a chain of steps, with intermediate results typically passed forward as SAS data sets.

Namespace Management
A DATA step can accommodate only one variable of a given name. In PROC SQL, multiple columns having the same name can be used successfully if they are differentiated by prefixes or qualifiers indicating their sources (see Section 4.1 for details). Of course, when results are **stored** in SAS data sets, duplicate names are not allowed.

Procedural versus Declarative
When you use a **procedural** language, you tell the computer what to do, not what to produce. The SAS DATA step language is procedural, though that fact is sometimes masked by all of the defaults and automatic behavior. With a **declarative** language, such as SQL, you tell the computer what to produce. The translation of such specifications into an operational plan is the responsibility of the software and is largely hidden from the user.

Row Order
Even though rows of data are supplied to PROC SQL in some order, and even though PROC SQL output is stored or displayed in some order (which the programmer can specify), SQL conceptually treats a given table or view as an unordered set of rows. Consequently, a query cannot explicitly or implicitly make reference to row ordering.

Bias for Normalized Data Structure
The subject of data normalization is a big one, and well beyond the scope of this book. For our purposes, we consider a table having sets of similar columns requiring parallel treatment to be “denormalized.” A table without such column sets, conversely, is termed “normalized,” and characteristically has more rows and fewer columns than its denormalized counterpart. For example, suppose you have data on exports by country and year. If you store your data in a long table with just three columns (one to identify the country, one to indicate the year, and one to report the value of exports), you have a normalized table. On the other hand, if you use a matrix structure with one row for each country and one column to identify the country plus one column for each year’s export data, your table is denormalized.

The DATA step is relatively neutral in supporting these alternative designs, with arrays and loops available to reference and process parallel columns. You could use an array and a loop to rather easily sum each year's exports. Many SAS procedures offer shortcuts for operating on sets of variables, so you could also use PROC MEANS with little difficulty to perform those aggregations. PROC SQL is different; it has nothing resembling arrays and loops. Consequently, SQL has a strong bias for normalized structures, and you will find SQL solutions much easier to develop if you organize your data accordingly. An example of such a normalized structure is presented in Section 12.3.

RDBMS Heritage

SQL comes to SAS from the world of relational database management systems, and that heritage shows in some ways. In particular, a lot of SQL statements are concerned with making changes to data **in place** (that is, inserting, deleting, or changing rows of data within a table, without replacing the table as a whole and without creating a new table coexistent with the original). SAS has such capabilities, but they are not widely used. SAS users are more likely to create new tables as they go along. We concentrate on doing things in that "SAS way" and confine our discussion of making changes in place to Chapter 9.

1.2 More about This Book

Before we take a detail-oriented look at PROC SQL in the following chapters, here are a few general how's and why's.

Purpose

The purpose of this book is to introduce PROC SQL to somewhat experienced SAS users. We start with the basics and then progress to more complex and specialized features.

We take almost every opportunity to illuminate SQL capabilities by demonstrating them together with more or less equivalent examples using the DATA step or SAS procedures other than PROC SQL. Because these non-SQL parallel techniques are not really the subject of the book, we don't explain them at length. However, we provide documentation references in those cases where the location of the relevant documentation might not be obvious, as well as references to *The Little SAS Book* (Fourth Edition). In addition, Chapter 14 provides something of a documentation "roadmap."

The SQL techniques presented in the book are specific to SAS in two ways. First, they occasionally use features of PROC SQL that extend beyond the requirements of the SQL standard. Second, they use language elements, such as SAS functions, that are not strictly part of SQL but rather are inherited from the SAS environment.

Perspective

Conceptually, and somewhat vaguely, we can divide the functionality of Base SAS software into four subsets. Moreover, we can visually suggest relationships among the four by positioning them in this diagram:

Beyond

Alternative SQL

Shared

Since it is our subject, let's start with the quadrant identified as "SQL." It comprises the statements and options supported by PROC SQL. That's pretty clear-cut, and there is just one caveat, which is that not everything coded within a PROC SQL step belongs to this subset of Base SAS. For example, a formula within an SQL statement might include a reference to the ABS (absolute value) function, which is not part of PROC SQL per se, but rather is borrowed from the Base SAS function collection.

"Alternative" refers to capabilities that are equivalent (substantially if not completely) to PROC SQL functionality but are found in the DATA step and a handful of "workhorse" procedures (primarily PRINT, SORT, MEANS/SUMMARY, FREQ, DATASETS, and FORMAT). It's important to understand that this quadrant does not encompass **all** features of the DATA step and the enumerated procedures; rather, it comprises only those features having parallels in PROC SQL.

The "Shared" quadrant represents things that make SAS work, or work better, and that support both SQL and Alternative usage. This quadrant includes

- the user environment and interfaces (such as the Display Manager)
- libraries and engines (so that PROC SQL can access any data set that other parts of SAS can access)
- most functions (but not call routines)
- formats and informats
- data set options and SAS system options

- global statements (such as TITLE)
- the SAS Macro Facility
- the Output Delivery System
- some utility procedures, such as PROC IMPORT, PROC EXPORT, and parts of PROC DATASETS

"Beyond" comprises all features of Base SAS that are beyond the practical limitations of PROC SQL (and thus, by implication, beyond the boundaries of the Alternative quadrant). For example, consider PROC CHART, which produces low-resolution graphs. Such output is outside the capability of PROC SQL, so PROC CHART belongs in the Beyond quadrant.

This four-way partitioning of Base SAS is strictly a conceptual exercise intended to explain the purpose of this book. The subdivisions have no operational significance and in fact are not mentioned after this chapter. They are also, admittedly, a bit vague and arbitrary. For example, consider that PROC MEANS or PROC SUMMARY can calculate both means and medians, whereas PROC SQL can calculate only the means. The implication is that a PROC MEANS step that calculates both means and medians has one foot in the Alternative quadrant and one in the Beyond quadrant. Similarly, PROC DATASETS can be used to manage both integrity constraints and audit trails (see Section 9.7), but PROC SQL can manage only the former. Thus, a PROC DATASETS step that deals with both of these tools belongs to both the Alternative and Shared subsets.

Presentation

The first half of the book deals exclusively with data retrieval queries. Chapters 2 and 3 address simple queries (defined as those that draw data from a single source). The next three chapters (4, 5, and 6) cover queries that tap into multiple sources.

Later chapters address a number of relatively specialized or advanced topics. Chapter 7 deals with options and Chapter 8 with the Macro Facility. In Chapter 9, we move beyond data retrieval to explain SQL tools for changing data. Chapter 10 takes on the subject of views, and Chapter 11 addresses SQL features for generating reports and concludes the presentation of SQL features.

The final few chapters supplement the earlier material. Chapter 12 presents examples that emphasize the use of SQL as a complement to other parts of SAS. Chapter 13 provides a short introduction to the important issue of performance tuning. Finally, Chapter 14 is a bit of an essay on the SAS documentation of SQL.

The book as a whole covers the major features of PROC SQL with one significant exception: interoperation with third-party RDBMS products. PROC SQL includes the Pass-Through Facility, a mechanism for sending SQL code to other vendors' RDBMS products to be processed and then receiving the results for use by SAS. Pass-through requires availability (licensing and installation) of the appropriate SAS/ACCESS product. The passed-through code must be written in the SQL "dialect" of the target RDBMS. Thus, working examples would depend on the choice of target system. For that reason, and because the focus of this book is on the Base SAS context, we do not discuss the Pass-Through Facility. SAS/ACCESS also permits SAS to operate with RDBMS servers more transparently, via LIBNAME statements. The behavior of PROC SQL code that exploits this feature depends on the nature of such code and the capabilities of the particular target RDBMS, and is also beyond the scope of the book.

Reference: For more information about PROC SQL interoperation with third-party RDBMS products, see *Base SAS 9.2 Procedures Guide*: Procedures: The SQL Procedure: Concepts: SQL Procedure: What Is the Pass-Through Facility? and Connecting to a DBMS Using the LIBNAME Statement.

This book does not attempt to repackage all of the information provided in the SAS documentation. For example, the use of PROC SQL options is explained (see Section 7.2), and a number of those options are identified and described there and elsewhere, but the coverage of such options is not comprehensive. The SAS documentation is available in four forms:

- hard copy
- online (Web pages at support.sas.com)
- PDF (Portable Document Format) files, available at support.sas.com
- locally installed Help files

All but the first are essentially free of cost. This book supports use of the documentation by providing numerous references. In addition, Chapter 14 is devoted to a discussion of the PROC SQL documentation.

The examples presented in the book were all set up and run using SAS 9.2 running on a Microsoft Windows XP host system. With the exception of a handful of LIBNAME statements, none of the code is specific to a host system, so it should be possible to run the examples on any SAS 9.2 platform and get the results shown. Because the PROC SQL feature set changed little between SAS 9.1.3 and SAS 9.2, there should be little difficulty replicating results with SAS 9.1.3.

All data libraries in the examples use the default native engine. The extent to which other engines can be successfully substituted will vary. Generally, there should not be problems in **reading** data from any engine. However, not all engines support all features for output.

The results displayed in the book were generated by running the examples in batch mode. However, there should be little or no effect on the results if Display Manager or other SAS user interfaces are used instead.

The examples attempt to simplify aspects of the code that are incidental to the subject at hand. So, for instance, nearly all tables in the examples are stored in the WORK library and denoted with one-part names (not prefixed with library references).

Reference: Read more about SAS libraries and library references in *SAS 9.2 Language Reference: Concepts*: SAS Files Concepts: SAS Data Libraries and in Section 1.11 of *The Little SAS Book* (Fourth Edition).

Now let's look at some of the visual properties of the examples. To illustrate, we'll borrow some bits and pieces from later chapters. Don't worry now about understanding the substance; explanations are provided in the original contexts.

First, here is a code specimen:

```
PROC SQL;
SELECT      fname, age
FROM        preteen;
QUIT;
```

Notice that it is indented and immediately follows a colon in the preceding text.

The use of uppercase and lowercase letters is significant in the examples. In code, names of SAS **language** elements (statements, clauses, options, functions, formats, and so on) appear in uppercase. The names of user-specified **data** elements (library references, data sets, variables, and so on) appear in lowercase or mixed case. This distinction applies only to code paragraphs (distinguishable by indention and use of a monospace font). In narrative paragraphs, **both** language and data element names are in uppercase; for example, we might refer to the FNAME column.

Notice that the preceding example includes a PROC SQL statement to launch the procedure and a QUIT statement to close the procedure. These statements are included because the SQL statements (such as the SELECT statement) work only if PROC SQL is already running. However, you should not interpret this to mean that each SQL statement must necessarily be in a separate procedure step. See Chapter 7 for more about PROC SQL session management.

The coded examples generate three types of results (although any particular example might involve only one or two):

- text appearing in the SAS log

- displayed output (that is, material that appears in the Output window—assuming that the code was run via the Display Manager)
- data sets produced or changed by the code

Let's look at these, starting with some log text:

```
NOTE: Table WORK.NEW created, with 7 rows and 5 columns.
```

The log text is set off with a ruled box rather than by indention. Now here is some displayed output:

```
  Median
    of 1
--------
     1.1
       6
     7.7
```

The presentation box is just like that used for log text. Distinctions between the two are either obvious or are noted in the narrative.

Examples in this book were run with the following SAS system options in effect: FORMCHAR="|----|+|---+=|-/\<>*" NOCENTER NODATE NOOVP.

Finally, let's see a result in the form of a data set. Of course, we are less interested in the hard-to-read bits and bytes of the data set as it is stored on disk than in a rendering of its content. Such a rendering is in Exhibit 1-1. Notice that this rendering is titled as an exhibit.

Exhibit 1-1 A table borrowed from a subsequent chapter

Sex	Youngest	Oldest	Avg_Height	Avg_Weight
F	11	12	55.8	70.7
M	11	12	59.7	98.9