

PROC SQL

Beyond the Basics Using SAS[®]

Third Edition



Kirk Paul Lafler

The correct bibliographic citation for this manual is as follows: Lafler, Kirk Paul. 2019. *PROC SQL: Beyond the Basics Using SAS®*, Third Edition. Cary, NC: SAS Institute Inc.

PROC SQL: Beyond the Basics Using SAS®, Third Edition

Copyright © 2019, SAS Institute Inc., Cary, NC, USA

978-1-64295-192-9(Hard cover)

978-1-63526-684-9 (Hardcopy)

978-1-63526-683-2 (Web PDF)

978-1-63526-681-8 (epub)

978-1-63526-682-5 (mobi)

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

March 2019

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 trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to

<http://support.sas.com/thirdpartylicenses>.

Contents

About This Book	vii
 Chapter 1: Designing Database Tables	 1
Introduction.....	1
Database Design.....	1
Column Names and Reserved Words	7
Data Integrity	8
Database Tables Used in This Book	8
Table Contents	11
Summary	20
 Chapter 2: Working with Data in PROC SQL.....	 21
Introduction.....	21
The SELECT Statement and Clauses	21
Overview of Data Types	23
SQL Operators, Functions, and Keywords	31
Dictionary Tables.....	66
Summary	82
 Chapter 3: Formatting Output.....	 83
Introduction.....	83
Formatting Output	83
Formatting Output with the Output Delivery System.....	102
Summary	108
 Chapter 4: Coding PROC SQL Logic.....	 109
Introduction.....	109
Conditional Logic.....	109
CASE Expressions	114
Interfacing PROC SQL with the Macro Language	139
Summary	151
 Chapter 5: Creating, Populating, and Deleting Tables	 153
Introduction.....	153
Creating Tables	154
Populating Tables.....	160
Integrity Constraints	177
Deleting Rows in a Table.....	189

Deleting Tables	190
Summary	193
Chapter 6: Modifying and Updating Tables and Indexes.....	195
Introduction	195
Modifying Tables	195
Indexes.....	207
Updating Data in a Table.....	218
Summary.....	219
Chapter 7: Coding Complex Queries	221
Introduction	222
Introducing Complex Queries.....	222
Joins.....	222
Why Joins Are Important.....	223
Cartesian Product Joins	229
Inner Joins.....	230
Outer Joins.....	240
Subqueries	246
Set Operations	256
Data Structure Transformations	265
Complex Query Applications	272
Summary.....	287
Chapter 8: Working with Views	289
Introduction	289
Views—Windows to Your Data	289
Eliminating Redundancy.....	299
Restricting Data Access—Security.....	299
Hiding Logic Complexities	300
Nesting Views	302
Updatable Views	304
Deleting Views	311
Summary.....	312
Chapter 9: Fuzzy Matching Programming	313
Introduction	313
Data Sets Used in Examples.....	314
6-Step Fuzzy Matching Process.....	316
Summary.....	342
Chapter 10: Data-driven Programming	343
Introduction	343
Programming Paradigms.....	343
SAS Metadata Sources	344

DICTIONARY Tables	350
CALL EXECUTE Routine	354
Custom-defined Formats	357
Macro Language	360
Summary	364
Chapter 11: Troubleshooting and Debugging.....	365
Introduction.....	365
The World of Bugs.....	365
The Debugging Process	366
Types of Problems	367
Troubleshooting and Debugging Techniques	368
Undocumented PROC SQL Options	382
Summary	389
Chapter 12: Tuning for Performance and Efficiency	391
Introduction.....	391
Understanding Performance Tuning.....	391
Sorting and Performance.....	392
User-Specified Sorting (SORTPGM= System Options).....	392
Grouping and Performance	393
Splitting Tables.....	393
Indexes and Performance	394
Reviewing CONTENTS Output and System Messages	395
Optimizing WHERE Clause Processing with Indexes	398
Summary	404
References.....	405

About This Book

What Does This Book Cover?

PROC SQL: Beyond the Basics Using SAS, Third Edition, is a step-by-step, example-driven guide that helps readers master the language of PROC SQL. Packed with analysis and examples illustrating an assortment of PROC SQL options, statements, and clauses, this book covers all the basics, but also offers extensive guidance on complex topics such as set operators and correlated subqueries.

The third edition explores new and powerful features in SAS® 9.4, including topics such as IFC and IFN functions, nearest neighbor processing, the HAVING clause, and indexes. It also features two completely new chapters on fuzzy matching and data-driven programming. Delving into the workings of PROC SQL with greater analysis and discussion, *PROC SQL: Beyond the Basic Using SAS, Third Edition*, examines a broad range of topics and provides greater detail about this powerful database language using discussion and numerous real-world examples.

Is This Book for You?

The intended audience for this book includes SAS users, programmers, business analysts, application and software developers, database analysts and administrators, help desk support staff, statisticians, IT support staff, and other professionals who want or need application-oriented information to extend their knowledge of PROC SQL beyond the basics.

This book offers readers under-the-hood, behind-the-scenes knowledge on how PROC SQL works and is the perfect tool for students pursuing an undergraduate or graduate information science, computer science, or cognitive science degree.

What Should You Know about the Examples?

This book includes tutorials for you to follow to gain hands-on experience with SAS.

Software Used to Develop the Book's Content

SAS® 9.4 was used to develop all content for this book.

Example Code and Data

You can access the example code and data for this book by linking to its author page at <https://support.sas.com/lafler>. Then, look for the cover thumbnail of this book, and select Example Code and Data to display the SAS programs that are included in this book.

SAS University Edition



This book is compatible with SAS University Edition. If you are using SAS University Edition, then begin here: <https://support.sas.com/ue-data>.

We Want to Hear from You

Do you have questions about a SAS Press book that you are reading? Contact us at saspress@sas.com.

SAS Press books are written *by* SAS Users *for* SAS Users. Please visit sas.com/books to sign up to request information on how to become a SAS Press author.

We welcome your participation in the development of new books and your feedback on SAS Press books that you are using. Please visit sas.com/books to sign up to review a book

Learn about new books and exclusive discounts. Sign up for our new books mailing list today at <https://support.sas.com/en/books/subscribe-books.html>.

Learn more about this author by visiting his author page at <http://support.sas.com/lafler>. There you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more.

Acknowledgments

This book was made possible because of the support and encouragement of many people. I would like to extend my sincerest thanks to each person who encouraged me to write this book. For, as G. B. Stern once said, “Silent gratitude isn’t very much use to anyone.” So from me to you, I want to thank everyone who had a hand in helping to make the first, second, and third editions possible.

To Sian Roberts for supporting my desire to introduce and develop new topics for the third edition. It was a distinct pleasure working with Sian where her encouragement, support, and coordination of the entire development process made this project possible.

To Suzanne Morgen, Development Editor, SAS Press for her guidance, feedback, and support, and for bringing clarity throughout the writing process.

To the technical reviewers who provided valuable feedback, suggestions, and technical accuracy on the new and revised chapters including the SAS code. Special thanks to Stephen Sloan, Vince DelGobbo, Leonid Batkhan, and Lewis Church for performing technical reviews of the third edition.

To the copy editors who made a difficult job look easy. A heart-felt thank you to Catherine Connolly who performed her magic identifying typos, and turning words, run-on sentences and, sometimes, long-winded paragraphs into coherent prose.

To Julie Palmieri, former Editor-in-Chief of SAS Press, for supporting my desire to develop a more comprehensive second edition. Her encouragement and support gave me the inspiration to see this project to completion.

To Stacey Hamilton at SAS Institute Inc. for her editorial assistance and coordinating the technical review process while I developed the second edition.

To Paul Kent at SAS Institute Inc. for his contagious enthusiasm, great examples, clear explanations, and mentorship with some of the fine points of PROC SQL and its capabilities over the years.

To David Baggett at SAS Institute Inc. for encouraging the idea of a book on PROC SQL and accepting the original first edition manuscript many years ago. His encouragement and support over the years has meant a great deal to me.

To Stephenie Joyner at SAS Institute Inc. for her support and encouragement during the development of the first edition and for coordinating the technical review process.

To Charles Edwin Shipp, Michael Raithel, Mary Rosenbloom, Richard La Valley, Clark Roberts, Roger Glaser, Stephen Sloan, Josh Horstman, MaryAnne DePesquo, Charu Shankar, Vince DelGobbo, Clarence Wm Jackson, Pablo Nogueras, Michael Johnston, Karen Walker, John Xu, Richann Watson, Deanna Schreiber-Gregory, Lex Jansen, Cynthia Zender, John Cohen, Sanjay Matange, Rick Langston, Sunil Gupta, Ron Cody, Russell Lavery, Mira Shapiro, Troy Martin Hughes, Ronald Fehd, Andrew Kuligowski, Russell Holmes, Peter Eberhardt, William Benjamin, Wei Cheng, Tyler Smith, and Toby Dunn for their friendship, support, and encouragement through the years.

To Art Carpenter of California Occidental Consultants for believing that a “beyond the basics” type of book on PROC SQL would be useful to SAS users.

To the many people at SAS Institute Inc. with whom I have developed so many friendships over the years. I’d like to express my thanks to each of you as well as all the knowledgeable people in SAS Technical Support. Thank you for developing and supporting a great product and enabling me to have a rewarding and enjoyable career for all these years.

To the SAS user group community around the world: You are the greatest group of professionals anywhere.

To the many teachers I have had in my life. Special thanks go to Lawrence Delk (6th grade); Mr. Almeida (12th grade); Professor Carl Kromp (Industrial Engineering); Joseph J. Moder, Ph.D. (Management Science); Charles N. Kurucz, Ph.D. (Management Science); John F. Stewart, Ph.D. (Computer Information Systems); Earl Wiener, Ph.D. (Management Science); Howard Seth Gitlow, Ph.D. (Management Science); Dean Paul K. Sugrue, Ph.D. (University of Miami School of Business); Edward K. Baker III, Ph.D. (Management Science); Robert T. Grauer, Ph.D. (Computer Information Systems); and Ulu (Rydacom) for sharing your knowledge and enthusiasm.

To the countless people I have worked with and the companies I have worked for – the experiences and memories have been invaluable.

To my mother, father, and brother for sharing life’s many lessons, experiences, and memories. Your love and encouragement through the years fueled my desire to learn, work hard, and experience life to the fullest.

Finally, to my wife, Darlynn, and son, Ryan, for your love, support, and sense of balance between family and work. I love you both so very much.

Thank you!

Chapter 1: Designing Database Tables

Introduction	1
Database Design.....	1
Conceptual View	2
Table Definitions	2
Redundant Information	3
Normalization	3
Normalization Strategies.....	4
Column Names and Reserved Words	7
ANSI SQL Reserved Words	7
SQL Code	7
Data Integrity.....	8
Referential Integrity	8
Database Tables Used in This Book.....	8
CUSTOMERS Table	8
INVENTORY Table	9
INVOICE Table.....	9
MANUFACTURERS Table	9
PRODUCTS Table	10
PURCHASES Table.....	10
Table Contents	11
The Database Structure	13
Sample Database Tables	14
Summary	20

Introduction

The area of database design is very important in relational processes. Much has been written on this subject, including entire textbooks and thousands of technical papers. No pretenses are made about the thoroughness of this very important subject in these pages. Rather, an attempt is made to provide a quick-start introduction for those readers who are unfamiliar with the issues and techniques of basic design principles. Readers needing more information are referred to the references listed in the back of this book. As you read this chapter, the following points should be kept in mind.

Database Design

Activities related to good database design require the identification of end-user requirements and involve defining the structure of data values on a physical level. Database design begins with a *conceptual view* of what is needed. The next step, called *logical design*, consists of developing a formal description of database entities and relationships to satisfy user requirements. Seldom does a database consist of a single table. Consequently, tables of

interrelated information are created to enable more complex and powerful operations on data. The final step, referred to as *physical design*, represents the process of achieving optimal performance and storage requirements of the logical database.

Conceptual View

The health and well-being of a database depends on its database design. A database must be in balance with all of its components (or optimized) to avoid performance and operation bottlenecks. Database design doesn't just happen and is not a process that occurs by chance. It involves planning, modeling, creating, monitoring, and adjusting to satisfy the endless assortment of user requirements without impeding resource requirements. Of central importance to database design is the process of planning. Planning is a valuable component that, when absent, causes a database to fall prey to a host of problems including poor performance and difficulty in operation. Database design consists of three distinct phases, as illustrated in Figure 1.1.

Figure 1.1: Three Distinct Phases of Database Design

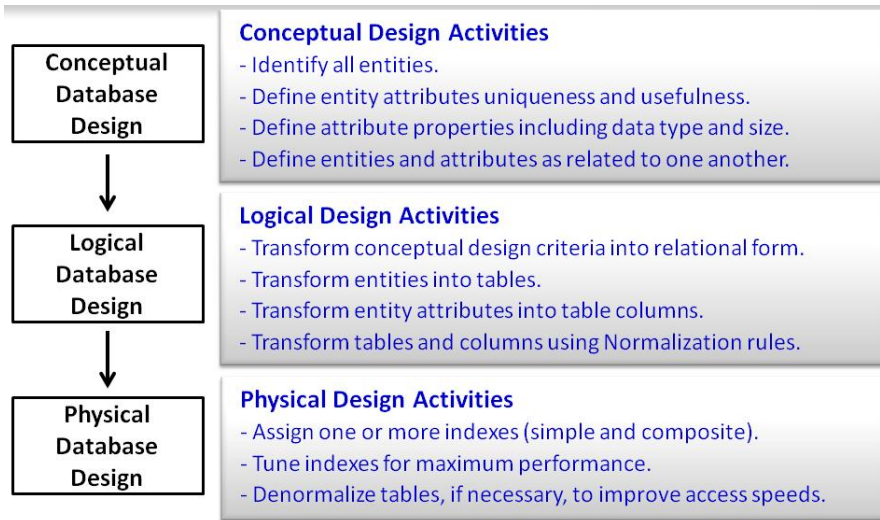


Table Definitions

PROC SQL uses a model of data that is conceptually stored as multisets rather than as physical files. A physical file consists of one or more records ordered sequentially or some other way. Programming languages such as COBOL and FORTRAN evolved to process files of this type by performing operations one record at a time. These languages were generally designed and used to mimic the way people process paper forms.

PROC SQL was designed to work with multisets of data. Multisets have no order, and members of a multiset are of the same type using a data structure known as a table. For classification purposes, a table is a base table consisting of zero or more rows and one or more columns, or a table is a virtual table (called a *view*), which can be used the same way that a table can be used (see Chapter 8, "Working with Views").

Redundant Information

One of the rules of good database design requires that data not be redundant or duplicated in the same database. The rationale for this conclusion originates from the belief that if data appears more than once in a database, then there is reason to believe that one of the pieces of data is likely to be in error. Furthermore, redundancy often leads to the following:

- Inconsistencies, because errors are more likely to result when facts are repeated.
- Update anomalies where the insertion, modification, or deletion of data may result in inconsistencies.

Another thing to watch for is the appearance of too many columns containing NULL values. When this occurs, the database is probably not designed properly. To alleviate potential table design issues, a process referred to as *normalizing* is performed. When properly done, this ensures the complete absence of redundant information in a table.

Normalization

The development of an optimal database design is an important element in the life cycle of a database. Not only is it critical for achieving maximum performance and flexibility while working with tables and data, it is essential to the organization of data by reducing or minimizing redundancy in one or more database tables. The process of table design is frequently referred to by database developers and administrators as *normalization*.

The normalization process is used for reducing redundancy in a database by converting complex data structures into simple data structures. It is carried out for the following reasons:

- To organize the data to save space and to eliminate any duplication or repetition of data.
- To enable simple retrieval of data to satisfy query and report requests.
- To simplify data manipulation requests such as data insertions, updates, and deletions.
- To reduce the impact associated with reorganizing or restructuring data as new application requirements arise.

The normalization process attempts to simplify the relationship between columns in a database by splitting larger multicolumn tables into two or more smaller tables containing fewer columns. The rationale for doing this is contained in a set of data design guidelines called *normal forms*. The guidelines provide designers with a set of rules for converting one or two large database tables containing numerous columns into a normalized database consisting of multiple tables and only those columns that should be included in each table. The normalization process consists of multiple steps with each succeeding step subscribing to the rules of the previous steps.

Normalization helps to ensure that a database does not contain redundant information in two or more of its tables. In an application, normalization prevents the destruction of data or the creation of incorrect data in a database. What this means is that information of fact is represented only once in a database, and any possibility of it appearing more than once is not, or should not be, allowed.

As database designers and analysts proceed through the normalization process, many are not satisfied unless a database design is carried out to at least third normal form (3NF). Joe Celko in his popular book *SQL for Smarties: Advanced SQL Programming* (Morgan Kaufman, 2014), describes 3NF this way: “Databases are considered to be in 3NF when a column is dependent on the key, the whole key, and nothing but the key.”

While the normalization guidelines are extremely useful, some database purists actually go to great lengths to remove any and all table redundancies even at the expense of performance. This is in direct contrast to other database experts who follow the guidelines less rigidly in an attempt to improve the performance of a database by only going as far as third normal form (or 3NF). Whatever your preference, you should keep this thought in mind as you normalize database tables. A fully normalized database often requires a greater number of joins and can adversely affect the speed of queries. Celko mentions that the process of joining multiple tables in a fully normalized database is costly, specifically affecting processing time and computer resources.

Normalization Strategies

After transforming entities and attributes from the conceptual design into a logical design, the tables and columns are created. This is when a process known as *normalization* occurs. Normalization refers to the process of making your database tables subscribe to certain rules. Many, if not most, database designers are satisfied when third normal form (3NF) is achieved and, for the objectives of this book, I will stop at 3NF, too. To help explain the various normalization steps, an example scenario follows.

First Normal Form (1NF)

First normal form (1NF) involves the elimination of data redundancy or repeating information from a table. A table is considered to be in first normal form when all of its columns describe the table completely and when each column in a row has only one value. A table satisfies 1NF when each column in a row has a single value and no repeating group information. Essentially, every table meets 1NF as long as an array, list, or other structure has not been defined. The following table illustrates a table satisfying the 1NF rule because it has only one value at each row-and-column intersection. The table is in ascending order by CUSTNUM and consists of customers and the purchases they made at an office supply store.

Table 1.1: First Normal Form (1NF) Table

CUSTNUM	CUSTNAME	CUSTCITY	ITEM	UNITS	UNITCOST	MANUCITY
1	Smith	San Diego	Chair	1	\$179.00	San Diego
1	Smith	San Diego	Pens	12	\$0.89	Los Angeles
1	Smith	San Diego	Paper	4	\$6.95	Washington
1	Smithe	San Diego	Stapler	1	\$8.95	Los Angeles
7	Lafler	Spring Valley	Mouse Pad	1	\$11.79	San Diego
7	Loffler	Spring Valley	Pens	24	\$1.59	Los Angeles
13	Thompson	Miami	Markers	.	\$0.99	Los Angeles

Second Normal Form (2NF)

Second normal form (2NF) addresses the relationships between sets of data. A table is said to be in second normal form when all the requirements of 1NF are met and a foreign key is used to link any data in one table which has relevance to another table. The very nature of leaving

a table in first normal form (1NF) may present problems due to the repetition of some information in the table. One noticeable problem is that Table 1.1 has repetitive information in it. Another problem is that there are misspellings in the customer name. Although repeating information may be permissible with hierarchical file structures and other legacy type file structures, it does pose a potential data consistency problem as it relates to relational data.

To describe how data consistency problems can occur, let's say that a customer takes a new job and moves to a new city. In changing the customer's city to the new location, it would be very easy to miss one or more occurrences of the customer's city resulting in a customer residing incorrectly in two different cities. Assuming that our table is only meant to track one unique customer per city, this would definitely be a data consistency problem. Essentially, second normal form (2NF) is important because it says that every non-key column must depend on the entire primary key.

Tables that subscribe to 2NF prevent the need to make changes in more than one place. What this means in normalization terms is that tables in 2NF have no partial key dependencies. As a result, our database that consists of a single table that satisfies 1NF will need to be split into two separate tables in order to subscribe to the 2NF rule. Each table would contain the CUSTNUM column to connect the two tables. Unlike the single table in 1NF, the tables in 2NF allow a customer's city to be easily changed whenever they move to another city because the CUSTCITY column only appears once. The tables in 2NF would be constructed as follows.

Table 1.2: CUSTOMERS Table

CUSTNUM	CUSTNAME	CUSTCITY
1	Smith	San Diego
1	Smithe	San Diego
7	Lafler	Spring Valley
13	Thompson	Miami

Table 1.3: PURCHASES Table

CUSTNUM	ITEM	UNITS	UNITCOST	MANUCITY
1	Chair	1	\$179.00	San Diego
1	Pens	12	\$0.89	Los Angeles
1	Paper	4	\$6.95	Washington
1	Stapler	1	\$8.95	Los Angeles
7	Mouse Pad	1	\$11.79	San Diego
7	Pens	24	\$1.59	Los Angeles
13	Markers	.	\$0.99	Los Angeles

Third Normal Form (3NF)

Referring to the two tables constructed according to the rules of 2NF, you may have noticed that the PURCHASES table contains a column called MANUCITY. The MANUCITY column stores the city where the product manufacturer is headquartered. Keeping this column in the PURCHASES table violates the third normal form (3NF) because MANUCITY does not provide factual information about the primary key column (CUSTNUM) in the PURCHASES table. Consequently, tables are considered to be in third normal form (3NF) when each column is dependent on the key, the whole key, and nothing but the key. The

tables in 3NF are constructed so the MANUCITY column would be in a table of its own as follows.

Table 1.4: CUSTOMERS Table

CUSTNUM	CUSTNAME	CUSTCITY
1	Smith	San Diego
1	Smithe	San Diego
7	Lafler	Spring Valley
13	Thompson	Miami

Table 1.5: PURCHASES Table

CUSTNUM	ITEM	UNITS	UNITCOST
1	Chair	1	\$179.00
1	Pens	12	\$0.89
1	Paper	4	\$6.95
1	Stapler	1	\$8.95
7	Mouse Pad	1	\$11.79
7	Pens	24	\$1.59
13	Markers	.	\$0.99

Table 1.6: MANUFACTURERS Table

MANUNUM	MANUCITY
101	San Diego
112	San Diego
210	Los Angeles
212	Los Angeles
213	Los Angeles
214	Los Angeles
401	Washington

Beyond Third Normal Form

In general, database designers are satisfied when their database tables subscribe to the rules in 3NF. But, it is not uncommon for others to normalize their database tables to fourth normal form (4NF) where independent one-to-many relationships between primary key and non-key columns are forbidden. Some database purists will even normalize to fifth normal form (5NF) where tables are split into the smallest pieces of information in an attempt to eliminate any and all table redundancies. Although constructing tables in 5NF may provide the greatest level of database integrity, it is neither practical nor desired by most database practitioners.

There is no absolute right or wrong reason for database designers to normalize beyond 3NF as long as they have considered all the performance issues that may arise by doing so. A common problem that occurs when database tables are normalized beyond 3NF is that a large number of small tables are generated. In these cases, an increase in time and computer resources frequently occurs because small tables must first be joined before a query, report, or statistic can be produced.

Column Names and Reserved Words

According to the American National Standards Institute (ANSI), SQL is the standard language used with relational database management systems. The ANSI Standard reserves a number of SQL keywords from being used as column names. The SAS SQL implementation is not as rigid, but users should be aware of what reserved words exist to prevent unexpected and unintended results during SQL processing. Column names should conform to proper SAS naming conventions (as described in the *SAS Language Reference*), and they should not conflict with certain reserved words found in the SQL language. The following list identifies the reserved words found in the ANSI SQL standard.

ANSI SQL Reserved Words

AS	INNER	OUTER
CASE	INTERSECT	RIGHT
EXCEPT	JOIN	UNION
FROM	LEFT	UPPER
FULL	LOWER	USER
GROUP	ON	WHEN
HAVING	ORDER	WHERE

You probably will not encounter too many conflicts between a column name and an SQL reserved word, but when you do you will need to follow a few simple rules to prevent processing errors from occurring. As was stated earlier, although PROC SQL's naming conventions are not as rigid as other vendor's implementations, care should still be exercised, in particular when PROC SQL code is transferred to other database environments expecting it to run error free. If a column name in an existing table conflicts with a reserved word, you have three options at your disposal:

1. Physically rename the column name in the table, as well as any references to the column.
2. Use the RENAME= data set option to rename the desired column in the current query.
3. Specify the PROC SQL option DQUOTE=ANSI, and surround the column name (reserved word) in double quotes, as illustrated below.

SQL Code

```
PROC SQL DQUOTE=ANSI;
  SELECT *
    FROM RESERVED_WORDS
   WHERE "WHERE"='EXAMPLE' ;
QUIT;
```

Data Integrity

Webster's New World Dictionary defines *integrity* as “the quality or state of being complete; perfect condition; reliable; soundness.” Data integrity is a critical element that every organization must promote and strive for. It is imperative that the data tables in a database environment be reliable, free of errors, and sound in every conceivable way. The existence of data errors, missing information, broken links, and other related problems in one or more tables can impact decision-making and information reporting activities resulting in a loss of confidence among users.

Applying a set of rules to the database structure and content can ensure the integrity of data resources. These rules consist of table and column constraints, and will be discussed in detail in Chapter 5, “Creating, Populating, and Deleting Tables.”

Referential Integrity

Referential integrity refers to the way in which database tables handle update and delete requests. Database tables frequently have a *primary key* where one or more columns have a unique value by which rows in a table can be identified and selected. Other tables may have one or more columns called a *foreign key* that are used to connect to some other table through its value. Database designers frequently apply rules to database tables to control what happens when a primary key value changes and its effect on one or more foreign key values in other tables. These referential integrity rules apply restrictions on the data that may be updated or deleted in tables.

Referential integrity ensures that rows in one table have corresponding rows in another table. This prevents lost linkages between data elements in one table and those of another enabling the integrity of data to always be maintained. Using the 3NF tables defined earlier, a foreign key called CUSTNUM can be defined in the PURCHASES table that corresponds to the primary key CUSTNUM column in the CUSTOMERS table. Users are referred to Chapter 5, “Creating, Populating, and Deleting Tables” for more details on assigning referential integrity constraints.

Database Tables Used in This Book

This section describes a database or library of tables that is used by an imaginary computer hardware and software wholesaler. The library consists of six tables: Customers, Inventory, Invoice, Manufacturers, Products, and Purchases. The examples used throughout this book are based on this library (database) of tables and are described and displayed below. An alphabetical description of each table used throughout this book appears below.

CUSTOMERS Table

The CUSTOMERS table contains customers that have purchased computer hardware and software products from a manufacturer. Each customer is uniquely identified with a customer number. A description of each column in the Customers table follows.

Table 1.7: Description of Columns in the Customers Table

CUSTNUM	Unique number identifying the customer.
CUSTNAME	Name of customer.
CUSTCITY	City where customer is located.

INVENTORY Table

The INVENTORY table contains customer inventory information consisting of computer hardware and software products. The Inventory table contains no historical data. As inventories are replenished, the old quantity is overwritten with the new quantity. A description of each column in the Inventory table follows.

Table 1.8: Description of Columns in the Inventory Table

PRODNUM	Unique number identifying product.
MANUNUM	Unique number identifying the manufacturer.
INVENQTY	Number of units of product in stock.
ORDDATE	Date product was last ordered.
INVENCST	Cost of inventory in customer's stock room.

INVOICE Table

The INVOICE table contains information about customers who purchased products. Each invoice is uniquely identified with an invoice number. A description of each column in the Invoice table follows.

Table 1.9: Description of Columns in the Invoice Table

INVNUM	Unique number identifying the invoice.
MANUNUM	Unique number identifying the manufacturer.
CUSTNUM	Customer number.
PRODNUM	Product number.
INVQTY	Number of units sold.
INVPRICE	Unit price.

MANUFACTURERS Table

The MANUFACTURERS table contains companies who make computer hardware and software products. Two companies cannot have the same name. No historical data is kept in this table. If a company is sold or stops making computer hardware or software, then the

manufacturer is dropped from the table. In the event that a manufacturer has an address change, the old address is overwritten with the new address. A description of each column in the Manufacturers table follows.

Table 1.10: Description of Columns in the Manufacturers Table

MANUNUM	Unique number identifying the manufacturer.
MANUNAME	Name of manufacturer.
MANUCITY	City where manufacturer is located.
MANUSTAT	State where manufacturer is located.

PRODUCTS Table

The PRODUCTS table contains computer hardware and software products offered for sale by the manufacturer. Each product is uniquely identified with a product number. A description of each column in the Products table follows.

Table 1.11: Description of Columns in the Products Table

PRODNUM	Unique number identifying the product.
PRODNAME	Name of product.
MANUNUM	Unique number identifying the manufacturer.
PRODTYPE	Type of product.
PRODCOST	Cost of product.

PURCHASES Table

The PURCHASES table contains computer hardware and software products purchased by customers. Each product is uniquely identified with a product number. A description of each column in the Purchases table follows.

Table 1.12: Description of Columns in the Purchases Table

CUSTNUM	Unique number identifying the customer.
ITEM	Name of product.
UNITS	Number of items purchased by customer.
UNITCOST	Cost of product.

Table Contents

An alphabetical list of tables, variables, and attributes for each table is displayed below.

Output 1.1: Customers CONTENTS Output

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
3	custcity	Char	20	Customer's Home City
2	custname	Char	25	Customer Name
1	custnum	Num	3	Customer Number

Output 1.2: Inventory CONTENTS Output

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
4	invcnst	Num	6	DOLLAR10.2		Inventory Cost
2	invenqty	Num	3			Inventory Quantity
5	manunum	Num	3			Manufacturer Number
3	orddate	Num	4	MMDDYY10.	MMDDYY10.	Date Inventory Last Ordered
1	prodnum	Num	3			Product Number

Output 1.3: Invoice CONTENTS Output

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
3	custnum	Num	3		Customer Number
1	invnum	Num	3		Invoice Number
5	invprice	Num	5	DOLLAR12.2	Invoice Unit Price
4	invqty	Num	3		Invoice Quantity - Units Sold
2	manunum	Num	3		Manufacturer Number
6	prodnum	Num	3		Product Number

Output 1.4: Manufacturers CONTENTS Output

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
3	manucity	Char	20	Manufacturer City
2	manuname	Char	25	Manufacturer Name
1	manunum	Num	3	Manufacturer Number
4	manustat	Char	2	Manufacturer State

Output 1.5: Products CONTENTS Output

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
3	manunum	Num	3		Manufacturer Number
5	prodcost	Num	5	DOLLAR9.2	Product Cost
2	prodname	Char	25		Product Name
1	prodnum	Num	3		Product Number
4	prodtype	Char	15		Product Type

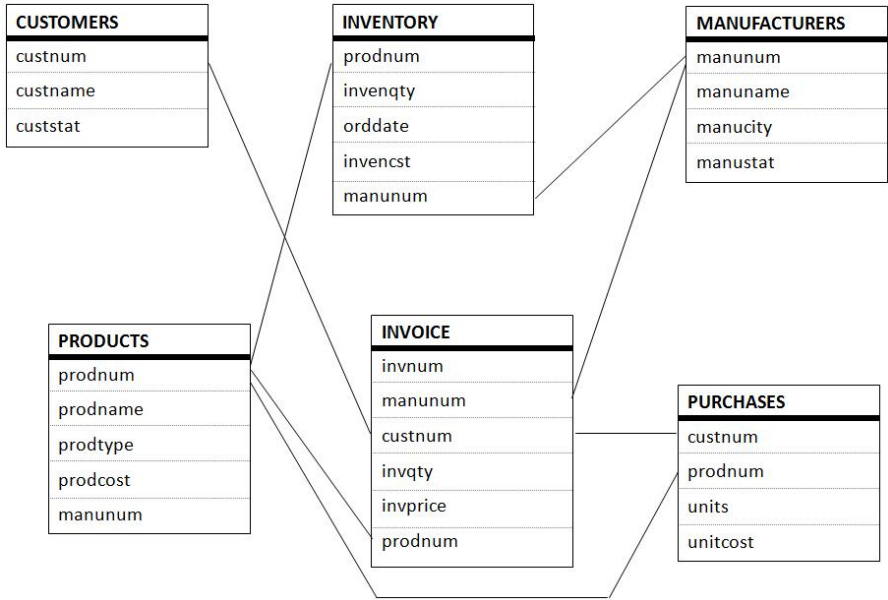
Output 1.6: Purchases CONTENTS Output

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
1	custnum	Num	4		Custnum
2	prodnum	Num	3		Prodnum
4	unitcost	Num	4	DOLLAR12.2	Unitcost
3	units	Num	3		Units

The Database Structure

The logical relationship between each table, and the columns common to each, appear below.

Figure 1.2. Logical Database Structure



Sample Database Tables

The following tables: Customers, Inventory, Manufacturers, Products, Invoice, and Purchases represent a relational database that will be illustrated in the examples in this book. These tables are small enough to follow easily, but complex enough to illustrate the power of SQL. The data contained in each table appears below.

Table 1.13: CUSTOMERS Table

Obs	custnum	custname	custcity
1	101	La Mesa Computer Land	La Mesa
2	201	Vista Tech Center	Vista
3	301	Coronado Internet Zone	Coronado
4	401	La Jolla Computing	La Jolla
5	501	Alpine Technical Center	Alpine
6	601	Oceanside Computer Land	Oceanside
7	701	San Diego Byte Store	San Diego
8	801	Jamul Hardware & Software	Jamul
9	901	Del Mar Tech Center	Del Mar
10	1001	Lakeside Software Center	Lakeside
11	1101	Bonsall Network Store	Bonsall
12	1201	Rancho Santa Fe Tech	Rancho Santa Fe
13	1301	Spring Valley Byte Center	Spring Valley
14	1401	Poway Central	Poway
15	1501	Valley Center Tech Center	Valley Center
16	1601	Fairbanks Tech USA	Fairbanks Ranch
17	1701	Blossom Valley Tech	Blossom Valley
18	1801	Chula Vista Networks	
N = 18			

Table 1.16: MANUFACTURERS Table

Obs	manunum	manuname	manucity	manustat
1	111	Cupid Computer	Houston	TX
2	210	Global Comm Corp	San Diego	CA
3	600	World Internet Corp	Miami	FL
4	120	Storage Devices Inc	San Mateo	CA
5	500	KPL Enterprises	San Diego	CA
6	700	San Diego PC Planet	San Diego	CA
N = 6				

Table 1.17: PRODUCTS Table

Obs	prodnum	prodname	manunum	prodtype	prodcost
1	1110	Dream Machine	111	Workstation	\$3,200.00
2	1200	Business Machine	120	Workstation	\$3,300.00
3	1700	Travel Laptop	170	Laptop	\$3,400.00
4	2101	Analog Cell Phone	210	Phone	\$35.00
5	2102	Digital Cell Phone	210	Phone	\$175.00
6	2200	Office Phone	220	Phone	\$130.00
7	5001	Spreadsheet Software	500	Software	\$299.00
8	5002	Database Software	500	Software	\$399.00
9	5003	Wordprocessor Software	500	Software	\$299.00
11	5004	Graphics Software	500	Software	\$299.00
N = 10					

Table 1.18: PURCHASES Table

Obs	custnum	prodnum	units	unitcost
1	1701	1110	1	\$3,200.00
2	101	5001	7	\$299.00
3	701	5001	11	\$299.00
4	701	5003	8	\$299.00
5	701	5002	4	\$399.00
6	701	5004	3	\$299.00
7	701	1700	2	\$3,400.00
8	701	1200	3	\$3,300.00
9	701	1110	2	\$3,200.00
10	1301	5001	3	\$299.00
11	1301	5003	5	\$299.00
12	1301	5002	2	\$399.00
13	901	1700	2	\$3,400.00
14	901	1200	3	\$3,300.00
15	901	1110	5	\$3,200.00
16	901	5001	9	\$299.00
17	901	5002	5	\$399.00
18	901	5003	8	\$299.00
19	901	5004	2	\$299.00
20	401	5001	11	\$299.00

21	401	5002	5	\$399.00
22	401	5003	7	\$299.00
23	401	5004	3	\$299.00
24	401	1700	3	\$3,400.00
25	401	1200	6	\$3,300.00
26	201	5001	6	\$299.00
27	201	5001	6	\$299.00
28	201	5003	9	\$299.00
29	201	5002	4	\$399.00
30	201	1700	3	\$3,400.00
31	901	5001	2	\$299.00
32	201	5001	2	\$299.00
33	201	2102	5	\$175.00
34	1101	2102	9	\$175.00
35	1301	2102	11	\$175.00
36	1401	2102	7	\$175.00
37	801	2102	5	\$175.00
38	501	2102	12	\$175.00
39	301	2102	8	\$175.00
40	1101	2200	3	\$130.00
41	101	2102	9	\$175.00

42	101	5003	3	\$299.00
43	101	5004	2	\$299.00
44	101	1200	3	\$3,300.00
45	101	1700	5	\$3,400.00
46	1301	1700	3	\$3,400.00
47	1601	1700	7	\$3,400.00
48	1801	1700	4	\$3,400.00
49	1001	1700	5	\$3,400.00
50	1101	1700	2	\$3,400.00
51	1201	1200	8	\$3,300.00
52	501	5001	3	\$299.00
53	501	5003	5	\$299.00
54	501	5004	1	\$299.00
55	501	1700	4	\$3,400.00
56	301	5001	6	\$299.00
57	501	2102	9	\$175.00
N = 57				

Summary

1. Good database design often improves the relative ease by which tables can be created and populated in a relational database and can be implemented into any database (see the “Conceptual View” section).
2. SQL was designed to work with sets of data and accesses a data structure known as a table or a “virtual” table, known as a view (see the “Table Definitions” section).
3. Achieving optimal design of a database means that the database contains little or no redundant information in two or more of its tables. This means that good database design calls for little or no replication of data (see the “Redundant Information” section).
4. Good database design avoids data redundancy, update anomalies, costly or inefficient processing, coding complexities, complex logical relationships, long application development times, and/or excessive storage requirements (see the “Normalization” section).
5. Design decisions made in one phase may involve making one or more tradeoffs in another phase (see the “Normalization” section).
6. A database in third normal form (3NF) is defined as a column that is dependent on the key, the whole key, and nothing but the key (see the “Normalization” section).