Chapter 1

Designing Database Tables

1.1 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 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.

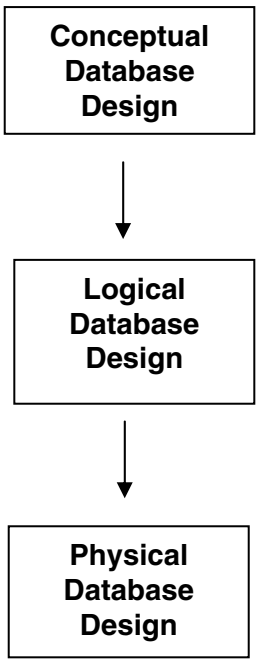
1.2 Database Design

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. In the final step, referred to as ***physical design***, the goal is to achieve optimal performance and efficient storage of the logical database.

1.2.1 Conceptual View

The health and well-being of a database depends on its database design. A database must be in balance (optimized) with all of its components to avoid performance and operation bottlenecks. Database design doesn't just happen. It involves planning, modeling, creating, monitoring, and adjusting to satisfy the endless assortment of user requirements without exhausting available resources. 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 below.



Conceptual Design Activities

1. Identify all entities.
2. Define entity attributes' uniqueness and usefulness.
3. Define attribute properties including data type, size, and whether null values can be accepted.
4. Define entities and attributes as related to one another.

Logical Design Activities

1. Transform conceptual design criteria into relational form.
2. Transform entities into tables.
3. Transform entity attributes into table columns.
4. Transform tables and columns using rules of functional dependencies and keys or normalization.

Physical Design Activities

1. Assign one or more indexes (simple and composite).
2. Tune indexes for maximum performance.
3. Denormalize tables, if necessary, to improve access speeds.

1.2.2 Table Definitions

PROC SQL uses a model of data stored as sets 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 sets of data. Sets have no order and members of a set are of the same type using a data structure known as a table. A table is either a base table consisting of zero or more rows with one or more columns or a virtual table called a view (see Chapter 8, "Working with Views").

1.2.3 Redundant Information

One of the rules of good database design is that data not be redundant or not be duplicated in the same database. The rationale for this is that if data appears more than once, then there is reason to believe that one of the pieces of data is likely to be in error. 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.

1.2.4 Normalization

Designing an optimal database design is an important element of database operations. It is also critical in achieving maximum performance and flexibility while working with tables and data. To minimize errors and duplication of data, database developers apply a concept called normalization to a logical database design.

The normalization process generally involves splitting larger multicolumn tables into two or more smaller tables containing fewer columns. The rationale for doing this is found 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 typically consists of no more than five steps with each succeeding step subscribing to the rules of the previous steps.

Normalizing a database helps to ensure that the database does not contain redundant information in two or more of its tables. 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 Kaufmann, 1999), describes 3NF this way: “Informally, all the non-key columns are determined by 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 the third step (or third normal form). Whatever your preference, you should keep this in mind as you normalize database tables. A fully normalized database often requires a greater number of joins and adversely affects the speed of queries. Celko mentions that the process of joining multiple tables is costly, specifically affecting processing time and computer resources.

1.2.5 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 will be given.

1.2.5.1 First Normal Form (1NF)

A table is considered to be in first normal form (1NF) 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 example 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.

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	$76.95	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

1.2.5.2 Second Normal Form (2NF)

The very nature of leaving a table in first normal form (1NF) may present problems because of the repetition of some information in the table as shown in the example above. Another problem is that there are misspellings in the customer names. 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, you might find it very easy to miss one or more occurrences 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 nonkey 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 consisting 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.

CUSTOMERS Table

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

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

1.2.6 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 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.

CUSTOMERS Table

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

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

MANUFACTURERS Table

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

1.2.7 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 nonkey 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.

1.3 Column Names and Reserved Words

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 vendors' 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 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;
```

1.4 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 affect 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."

1.4.1 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 is 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 restrict 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.

1.5 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 manufacturer. The library consists of six tables: customer, 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.

1.5.1 CUSTOMERS Table

The CUSTOMERS table contains data on 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.

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

1.5.2 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.

INVENTORY	
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

1.5.3 INVOICE Table

The INVOICE table contains information about customer purchases. Each invoice is uniquely identified with an invoice number. A description of each column in the invoice table follows:

INVOICE	
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

1.5.4 MANUFACTURERS Table

The MANUFACTURERS table contains data about companies that 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 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.

MANUFACTURERS

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

1.5.5 PRODUCTS Table

The PRODUCTS table contains data about 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.

PRODUCTS

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

1.5.6 PURCHASES Table

The PURCHASES table contains information about 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.

PURCHASES	
CUSTNUM	Unique number identifying the product
ITEM	Name of product
UNITS	Unique number identifying the manufacturer
UNITCOST	Cost of product

1.6 Table Contents

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

Customers CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Label
custcity	Char	20	25	Customer's Home City
custname	Char	25	0	Customer Name
custnum	Num	3	45	Customer Number

Inventory CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Format	Informat	Label
invencst	Num	6	10	DOLLAR10.2		Inventory Cost
invenqty	Num	3	7			Inventory Quantity
manunum	Num	3	16			Manufacturer Number
orddate	Num	4	0	MMDDYY10.	MMDDYY10	Date Inventory Last Ordered
prodnum	Num	3	4			Product Number

Invoice CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Format	Label
custnum	Num	3	6		Customer Number
invnum	Num	3	0		Invoice Number
invprice	Num	5	12	DOLLAR12.2	Invoice Unit Price
invqty	Num	3	9		Invoice Quantity - Units Sold
manunum	Num	3	3		Manufacturer Number
prodnum	Num	3	17		Product Number

Manufacturers CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Label
manucity	Char	20	25	Manufacturer City
manuname	Char	25	0	Manufacturer Name
manunum	Num	3	47	Manufacturer Number
manustat	Char	2	45	Manufacturer State

Products CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Format	Label
manunum	Num	3	40		Manufacturer Number
prodcost	Num	5	43	DOLLAR9.2	Product Cost
prodname	Char	25	0		Product Name
prodnum	Num	3	48		Product Number
prodtype	Char	15	25		Product Type

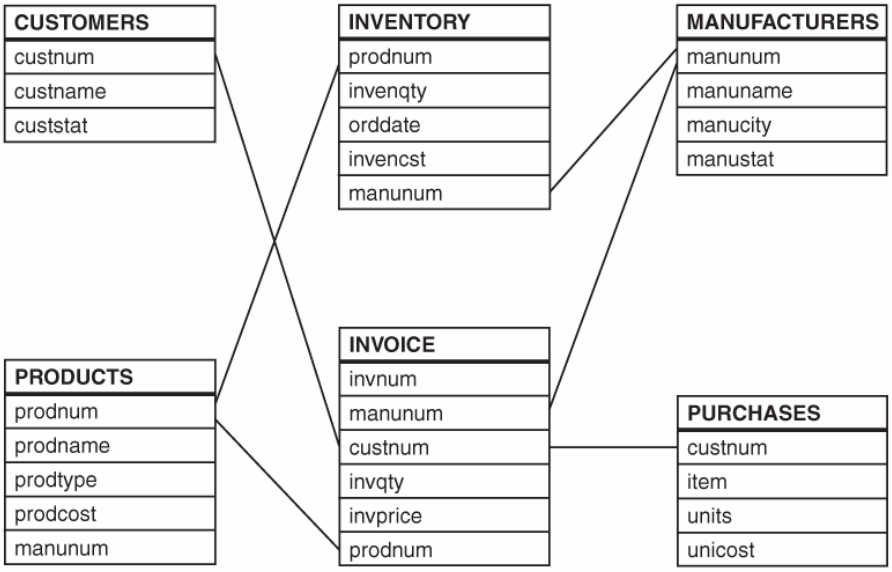
Purchases CONTENTS Output

-----Alphabetic List of Variables and Attributes-----

Variable	Type	Len	Pos	Format
custnum	Num	4	0	
item	Char	10	8	
unitcost	Num	4	4	DOLLAR12.2
units	Num	3	18	

1.6.1 The Database Structure

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



1.6.2 Sample Database Tables

The six tables (named above) 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.

CUSTOMERS Table

custnum	custname	custcity
101	La Mesa Computer Land	La Mesa
201	Vista Tech Center	Vista
301	Coronado Internet Zone	Coronado
401	La Jolla Computing	La Jolla
501	Alpine Technical Center	Alpine
601	Oceanside Computer Land	Oceanside
701	San Diego Byte Store	San Diego
801	Jamul Hardware & Software	Jamul
901	Del Mar Tech Center	Del Mar
1001	Lakeside Software Center	Lakeside
1101	Bonsall Network Store	Bonsall
1201	Rancho Santa Fe Tech	Rancho Santa Fe
1301	Spring Valley Byte Center	Spring Valley
1401	Poway Central	Poway
1501	Valley Center Tech Center	Valley Center
1601	Fairbanks Tech USA	Fairbanks Ranch

(*continued on next page*)

custnum	custname	custcity
1701	Blossom Valley Tech	Blossom Valley
1801	Chula Vista Networks	

N = 18

INVENTORY Table

prodnum	invenqty	orddate	invencst	manunum
1110	20	09/01/2000	$45,000.00	111
1700	10	08/15/2000	$28,000.00	170
5001	5	08/15/2000	$1,000.00	500
5002	3	08/15/2000	$900.00	500
5003	10	08/15/2000	$2,000.00	500
5004	20	09/01/2000	$1,400.00	500
5001	2	09/01/2000	$1,200.00	600

N = 7

INVOICE Table

invnum	manunum	custnum	invqty	invprice	prodnum
1001	500	201	5	$1,495.00	5001
1002	600	1301	2	$1,598.00	6001
1003	210	101	7	$245.00	2101
1004	111	501	3	$9,600.00	1110
1005	500	801	2	$798.00	5002
1006	500	901	4	$396.00	6000
1007	500	401	7	$23,100.00	1200

N = 7

MANUFACTURERS Table

manunum	manuname	manucity	manustat
111	Cupid Computer	Houston	TX
210	KPL Enterprises	San Diego	CA
600	World Internet Corp	Miami	FL
120	Storage Devices Inc	San Mateo	CA
500	Global Software	San Diego	CA
700	San Diego PC Planet	San Diego	CA

N = 6

PRODUCTS Table

prodnum	prodname	manunum	prodtype	prodcost
1110	Dream Machine	111	Workstation	$3,200.00
1200	Business Machine	120	Workstation	$3,300.00
1700	Travel Laptop	170	Laptop	$3,400.00
2101	Analog Cell Phone	210	Phone	$35.00
2102	Digital Cell Phone	210	Phone	$175.00
2200	Office Phone	220	Phone	$130.00
5001	Spreadsheet Software	500	Software	$299.00
5002	Database Software	500	Software	$399.00
5003	Wordprocessor Software	500	Software	$299.00
5004	Graphics Software	500	Software	$299.00

N=10

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

(*continued on next page*)

custnum	item	units	unitcost
7	Pens	24	$1.59
13	Markers	.	$0.99
N=7			

1.7 Summary

1. Poor database design is often attributed to the relative ease by which tables can be created and populated in a relational database. By adhering to certain rules, good design can be structured into almost any database (see section 1.2.1).

2. SQL was designed to work with sets of data and accesses a data structure known as a table (see section 1.2.2).

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 section 1.2.3).

4. Poor database design can result in costly or inefficient processing, coding complexities, complex logical relationships, long application development times, or excessive storage requirements (see section 1.2.4).

5. Design decisions made in one phase may involve making one or more tradeoffs in another phase (see section 1.2.4).

6. A database in third normal form (3NF) is where a column is “dependent on the key, the whole key, and nothing but the key” (see section 1.2.4).