

INFORMIX-SQL

Relational Database Management System

USER GUIDE

Copyright © 1981-1987
Informix Software, Inc.
July 1987

INFORMIX-SQL
Version 2.10

Part No. 200-404-1015-0
Rev. A
PRINTED IN U.S.A.

THE INFORMIX SOFTWARE AND USER MANUAL ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE INFORMIX SOFTWARE AND USER MANUAL IS WITH YOU. SHOULD THE INFORMIX SOFTWARE AND USER MANUAL PROVE DEFECTIVE, YOU (AND NOT INFORMIX OR ANY AUTHORIZED REPRESENTATIVE OF INFORMIX) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. IN NO EVENT WILL INFORMIX BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH INFORMIX SOFTWARE OR USER MANUAL, EVEN IF INFORMIX OR AN AUTHORIZED REPRESENTATIVE OF INFORMIX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. IN ADDITION, INFORMIX SHALL NOT BE LIABLE FOR ANY CLAIM ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH INFORMIX SOFTWARE OR USER MANUAL BASED UPON STRICT LIABILITY OR INFORMIX'S NEGLIGENCE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

All rights reserved. No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without permission of the publisher.

Published by:
Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025

INFORMIX is a registered trademark of Informix Software, Inc. **RDSQL**, **C-ISAM**, and **File-it!** are trademarks of Informix Software, Inc.

UNIX is a trademark of AT&T.

MS is a trademark of Microsoft Corporation.

VMS and VT are trademarks of Digital Equipment Corporation.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software Clause at 52.227-7013 (and any other applicable license provisions set forth in the Government contract).

Copyright © 1981-1987 by Informix Software, Inc.

NOTE: ANY REFERENCES IN DOCUMENTATION TO "RELATIONAL DATABASE, INC." OR "RDS" SHALL MEAN INFORMIX SOFTWARE, INC.

Table of Contents

Preface

About This Guide.....	5
Chapter Summary	6
Operating System References	7
Conventions Used in This Guide.....	7
Related Reading.....	9
Preparing to Use INFORMIX-SQL.....	10
Setting Up the Operating Environment	11
Using Your Terminal	15
The Demonstration Database	17
Restricting Database Access on Multi-User Systems.....	19

Chapter 1. Introducing INFORMIX-SQL

Chapter Overview.....	1-5
What Is INFORMIX-SQL?	1-6
What Is a Database?.....	1-7
How INFORMIX-SQL Stores Data.....	1-8
Retrieving Data from the Database	1-11
Connecting Tables through Column Joins	1-13
Getting Started with INFORMIX-SQL	1-16
Accessing INFORMIX-SQL on DOS Systems	1-16
Loading the Database Agent.....	1-17
Entering INFORMIX-SQL	1-17
The INFORMIX-SQL Main Menu.....	1-18
The Current Option	1-19
The Message Line	1-19
Moving the Highlight.....	1-20
Selecting an Option.....	1-20
What the Main Menu Options Mean.....	1-20
Using the HELP Menu.....	1-23
The INFORMIX-SQL Facilities	1-24
Creating Tables with the Schema Editor	1-24

The Screen Form Features.....	1-26
The Report Writing Facilities.....	1-29
The RDSQL Structured Query Language	1-32
The User-menu System	1-34
A Complete Database Management System	1-35
Chapter Summary	1-36

Chapter 2. Creating a Database

Chapter Overview.....	2-5
Creating the Demonstration Database.....	2-5
Two Ways to Create Databases and Tables.....	2-6
Using the Main Menu to Create Databases and Tables	2-6
Using RDSQL to Create Databases and Tables	2-6
Accessing INFORMIX-SQL on DOS Systems	2-7
Loading the Database Agent.....	2-7
Entering INFORMIX-SQL	2-8
Accessing INFORMIX-SQL on UNIX Systems.....	2-9
Using the Main Menu to Create a Database	2-10
Naming the New Database.....	2-11
The Current Database	2-12
System Files that Make Up a Database	2-13
Selecting a Different Current Database	2-14
Leaving a Current Menu or Screen.....	2-15
Updating an Existing INFORMIX-SQL Database	2-15
Using the Main Menu to Create a Table	2-16
Naming the New Table.....	2-17
Using the CREATE TABLE Menu	2-18
An Overview of Building a Table	2-18
Naming the Columns	2-20
Assigning Data Types	2-21
Assigning a Data Type to the Column.....	2-25
Creating Indexes	2-27
NULL Values	2-29
Adding the Remaining Columns to the Table	2-30
Exiting the Schema Editor.....	2-31
System Files	2-31
Changing the Table Specifications.....	2-30
Chapter Summary	2-31

Chapter 3. Entering Data

Chapter Overview	3-5
What Is a Screen Form?	3-6
Screen Forms in the Demonstration Database	3-6
What Is PERFORM?	3-8
Including Data from Multiple Tables	3-8
Each Table Can Contribute to Several Forms	3-9
Using a Screen Form	3-9
Choosing the Form You Want	3-10
The PERFORM Screen	3-11
Information Lines	3-12
Screen Form	3-13
Status Lines	3-13
The PERFORM Menu Options	3-14
Adding Data to a Database	3-16
Entering Data in the Fields	3-18
Confirming Your Entry	3-20
Data Checking in PERFORM	3-20
Special Functions	3-21
Positioning the Cursor	3-21
Field Editing	3-22
Storing the Row	3-24
Changing Existing Data	3-25
Removing a Row	3-26
Exiting PERFORM	3-27
Chapter Summary	3-28

Chapter 4. Querying a Database

Chapter Overview	4-5
What Is a Database Query?	4-6
Searching for an Exact Match	4-7
Searching for a Group of Rows	4-9
Finding All the Rows in a Table	4-12
How Database Queries Work	4-13
Searching with Relational Operators	4-14
Using Wildcard Characters	4-16
Searching for a Range of Values	4-17
Searching for the Highest and Lowest Values	4-18

Searching for Alternate Values	4-19
Query Operators and Short Fields.....	4-19
Searching in SMALLFLOAT and FLOAT Fields.....	4-20
Example Database Queries.....	4-21
Moving from Screen to Screen	4-22
Saving the Results of a Query	4-23
Chapter Summary	4-27

Chapter 5. Using Multiple-Table Forms

Chapter Overview.....	5-5
What Is a Multiple-Table Form?	5-6
The Active Table.....	5-7
Changing the Active Table	5-7
Example	5-8
Join Fields.....	5-11
The orderform Form	5-12
Verify Joins	5-14
Lookup Joins.....	5-15
The Master-Detail Relationship.....	5-16
Selecting the Detail Table	5-16
Returning to the Master Table.....	5-19
Multiple Tables Mean Multiple Current Lists	5-20
The Current Option	5-20
Chapter Summary	5-21

Chapter 6. Creating Your Own Forms

Chapter Overview.....	6-5
What Is a Form Specification?	6-6
Default Form Specifications.....	6-6
How to Create a Default Form	6-7
How to Modify an Existing Form	6-7
How to Create a Form.....	6-8
Naming the Form	6-9
Choosing the Tables to Include in the Form	6-10
Compiling the Default Form Specification.....	6-11
What a Default Form Looks Like	6-13
An Example Default Form	6-14

How to Modify an Existing Form	6-15
What the Form Specification Contains	6-16
The Example Form Specification	6-17
The customer Form Specification	6-20
Comparing the custom and customer Forms	6-21
Using FORMBUILD Attributes	6-24
How to Enter Attributes.....	6-25
Displaying Comments on the Screen	6-26
Changing Data from Lowercase to Uppercase	6-27
Changing Default Field Values	6-28
Requiring Input	6-30
Specifying Acceptable Values.....	6-31
Verifying Input	6-33
Joining Fields.....	6-33
Instructions Section	6-40
Chapter Summary	6-41

Chapter 7. Using RDSQL

Chapter Overview.....	7-5
Examples Used in This Chapter.....	7-5
What Is RDSQL?	7-6
An Interactive Query Language.....	7-6
Using RDSQL Statements	7-6
Accessing RDSQL	7-7
RDSQL Options	7-8
RDSQL SYNTAX Menu.....	7-9
Differences between RDSQL and PERFORM	7-9
How to Create a Table with RDSQL.....	7-11
Entering an RDSQL Statement.....	7-12
Using the RDSQL Editor	7-12
Using the Use-Editor Option	7-14
Naming the Table.....	7-15
Naming the Columns	7-15
Assigning Data Types	7-16
Running an RDSQL Statement	7-18
If There Are Errors.....	7-19
Saving the Current Statements	7-20
The CREATE INDEX Statement	7-21
Naming the Index.....	7-21

Specifying a Table	7-22
Specifying Index Columns	7-22
Clustered Indexes	7-23
Preventing Duplicate Entries.....	7-23
Ascending and Descending Indexes.....	7-24
Using a Command File	7-25
Database Privileges on Multiuser Systems	7-26
Querying a Database	7-28
Entering a Statement.....	7-28
Formatting RDSQL Statements.....	7-29
Searching for Rows and Columns: The SELECT Statement.....	7-30
Searching for All Rows and All Columns	7-31
Searching for Specific Columns	7-32
Searching for Specific Rows.....	7-38
Sorting Query Results	7-43
Queries with More than One Table.....	7-46
Sending Query Results to the Printer or a File.....	7-51
Sending Results to the Printer	7-51
Sending Results to a File.....	7-52
Modifying Data with RDSQL.....	7-54
Adding Rows to a Table	7-54
Changing the Data in a Table.....	7-57
Deleting Rows from a Table.....	7-59
Displaying Table Information.....	7-60
Chapter Summary	7-62

Chapter 8. Creating and Printing Reports

Chapter Overview	8-5
What Is a Report Writer?	8-6
Steps in Creating a Report	8-7
Using the REPORT Menu	8-8
A Sample Specification and Report	8-8
Sections of a Specification.....	8-11
Creating a Report Specification	8-14
Generating a Default Report Specification	8-15
Naming the Report.....	8-15
Compiling the Default Report Specification	8-16
Running the Report	8-17

Modifying a Report.....	8-20
Saving the Report Specification	8-21
Compiling a Report Specification	8-23
If the Compilation Is Unsuccessful	8-23
What a Report Specification Contains	8-26
Selecting the Data for a Report	8-28
SELECT and FORMAT Work Together	8-28
Selecting All Rows and Columns of a Table	8-29
Selecting Data from More than One Table	8-30
Basic Report Formatting.....	8-31
The OUTPUT Section.....	8-31
The FORMAT Section	8-32
Creating Page Headers and Trailers	8-33
Printing Rows of Data	8-37
Grouping Data	8-40
Calculations in a Report.....	8-43
Arithmetic Operators	8-43
Controlling the Appearance of Numbers	8-45
Math Functions	8-48
Group Functions.....	8-54
Displaying a Report	8-57
Printing the Report on a Printer.....	8-57
Writing the Report to a File	8-58
Piping the Report to Another Program.....	8-59
Chapter Summary.....	8-60

Chapter 9. Database Structure and Integrity

Chapter Overview.....	9-5
Changing the Structure of a Database.....	9-6
Removing a Table.....	9-6
Removing an Index	9-7
Renaming a Table	9-7
Removing a Database.....	9-8
Changing the Structure of a Table.....	9-9
A Look at the customer Table	9-10
Using RDSQL	9-10
Using the Schema Editor.....	9-16
Transactions	9-19
Audit Trails.....	9-20

Comparing Audit Trails and the Transaction Log.....	9-20
Views.....	9-21
Data Types.....	9-22
CHAR Columns	9-22
Numeric Columns.....	9-23
DATE Columns	9-26
MONEY Columns	9-26
Column and Row Lengths.....	9-27
Creating Indexes.....	9-29
When to Index Columns.....	9-29
Columns You Can Index.....	9-30
Columns You Should Not Index.....	9-32
Preventing Duplicate Entries.....	9-32
Ascending and Descending Indexes.....	9-33
Clustered Indexes	9-33
Chapter Summary	9-34

Appendix A. The stores Database

Appendix B. Environment Variables

Appendix C. INFORMIX-SQL Reserved Words

Index

new Features in INFORMIX-SQL Version 2.10

The following enhancements have been made to INFORMIX-SQL Version 2.10.

Auto-Indexing (speeds unindexed queries)

If you execute a SELECT statement or create a screen form that includes a join between two tables and there are no indexes on the joined columns, RDSQL now creates a temporary index on the table with the larger number of rows. This enhancement results in a dramatic improvement in the speed of unindexed queries.

Cluster Indexing (physically orders data)

Since both UNIX and DOS extract information from the disk in blocks, the more rows that are physically on the same block and that are already in the same order as an index, the faster an indexed retrieval proceeds. A cluster index causes the physical order in the table to be the same as the order of an index.

Wait for Locked Row (waits on any locked row)

The new SET LOCK MODE statement can be used to determine whether RDSQL waits for a locked row to become unlocked.

UPDATE Extensions (allow greater flexibility)

The syntax of the UPDATE statement has been expanded to allow greater flexibility when specifying the list of column-names and values.

New PERFORM Menu (increases ease of use)

The PERFORM Menu now appears and functions in a manner consistent with the other INFORMIX-SQL menus. The menu options occupy the top line only; a description of the option appears on the second (message) line of the two-page menu.

New PERFORM Keywords (prevent deletions)

You can use the BEFORE REMOVE operation to cause PERFORM to take one or more actions before removing a row from a database table. ABORT is a new option.

New PERFORM Operators (expand performance)

The pipe sign (|) can be used in queries to signify “or,” and the question mark (?) matches a single character. In addition, the >> and << symbols have been modified to work in unindexed columns.

Views in PERFORM (improve data integrity)

Screen forms can now include views.

New Command Line Syntax (reduces access time)

The **INFORMIX-SQL** modules can now be accessed directly from the operating system command line using a shortened version of the **INFORMIX-SQL** Main Menu options. In addition, a list of compiled form or report specifications can be included on the command line.

Redesigned User-Menu (streamlines operations)

The user-menu screen has been redesigned to reduce the work required to build a custom user-menu.

The dbschema Utility (replicates a database)

You can use the **dbschema** utility to quickly produce an **RDSQL** command file containing the **CREATE TABLE**, **CREATE INDEX**, **CREATE VIEW**, **CREATE SYNONYM**, and **GRANT** statements required to replicate an entire database or a selected table.

Preface

Preface Table of Contents

About This Guide	5
Chapter Summary.....	6
Operating System References.....	7
Conventions Used in This Guide	7
Syntax.....	8
Related Reading	9
Preparing to Use INFORMIX-SQL	10
Setting Up the Operating Environment.....	11
Setting Environment Variables on UNIX Systems.....	11
Setting Environment Variables on DOS Systems.....	13
Using Your Terminal.....	15
The Demonstration Database.....	17
Creating the Demonstration Database	17
Restoring the Demonstration Database	18
Restricting Database Access on Multi-User Systems.....	19

About This Guide

The *INFORMIX-SQL User Guide* and its companion volume, the *INFORMIX-SQL Reference Manual*, comprise the comprehensive documentation for the INFORMIX-SQL relational database management system.

The *INFORMIX-SQL User Guide* is an introduction to INFORMIX-SQL. You do not need database management experience or familiarity with basic database management concepts to use this manual. The *INFORMIX-SQL User Guide* includes general information about database systems and leads you through the steps necessary to create a database, enter and access database information, and produce printed reports. The first chapter covers important basic information about database systems. Subsequent chapters contain instructions and illustrations that show you how to take advantage of many of the powerful INFORMIX-SQL features. Each chapter also tells you where to look in the *INFORMIX-SQL Reference Manual* for information about advanced features.

The *INFORMIX-SQL Reference Manual* is a complete reference to the programs that make up INFORMIX-SQL. It contains information about everything you can do with INFORMIX-SQL, and is organized by program name. Once you are familiar with INFORMIX-SQL basics, you can use the *INFORMIX-SQL Reference Manual* to learn about advanced features and to quickly locate specific information.

The manual includes two additional documents:

- *Getting Started: A Guide to Products and Services* contains customer registration and warranty information, along with descriptions of other RDS products.
- The *INFORMIX-SQL Quick Reference* card displays syntax statements for the RDSQL query language, the PERFORM screen form generator, and the ACE report writer.

Chapter Summary

Each chapter in this book begins with a section describing chapter contents. This section includes a paragraph to help you identify topics appropriate for your needs. The rest of the chapter explains each topic area and demonstrates its use in **INFORMIX-SQL**. A concluding summary highlights the most important points of the chapter.

The *INFORMIX-SQL User Guide* includes the following chapters and appendixes:

- **Chapter 1** introduces important concepts about database management systems.
- **Chapter 2** describes how to create **INFORMIX-SQL** databases, tables, and indexes.
- **Chapter 3** explains how to enter data into database tables, change data, and delete data.
- **Chapter 4** describes how you can use screen forms to query data.
- **Chapter 5** explains how you can use screen forms that include multiple database tables.
- **Chapter 6** describes how you can create your own custom screen forms.
- **Chapter 7** describes the **RDSQL** query language, which you can use to enter, alter, delete, or retrieve data from the database.
- **Chapter 8** explains how to create, display, and print reports.
- **Chapter 9** describes how to alter the structure of databases and tables, and how to maintain data integrity.
- **Appendix A** describes the **stores** demonstration database.

- **Appendix B** describes how to use *environment variables* to customize **INFORMIX-SQL** features.
- **Appendix C** lists the **INFORMIX-SQL** reserved words.

Operating System References

INFORMIX-SQL is available for both UNIX and DOS (PC-DOS and MS-DOS) operating systems. The few differences in performance and screen display between the two systems are noted whenever they occur.

For ease of reading, PC-DOS and MS-DOS both are referred to as “DOS” systems.

Conventions Used in This Guide

The *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual* use a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

When new terms are introduced, they are printed in italics, *like this*. The documentation also includes illustrations that show what you see on the screen as you use **INFORMIX-SQL**. Information that **INFORMIX-SQL** displays and information that you enter are printed in a computer typeface, like this.

Syntax statements describe the format of **INFORMIX-SQL** statements or commands, including alternate forms of a statement, required and optional parts of the statement, and so forth. Syntax statements have their own rules; these are defined in detail in the following section. Briefly, all *keywords* are shown in uppercase letters for ease of identification, even though you need not enter them that way. Words for which you must supply values are in italics. When keywords or values are enclosed in square brackets ([]), they are optional parts of the statement (unless otherwise indicated).

Syntax

The following notational conventions are used in syntax statements that appear in this book, as well as in the *INFORMIX-SQL Reference Manual*:

ABC Any term in the syntax in uppercase letters is a keyword. Enter it exactly as shown, disregarding case. For example,

```
CREATE TABLE table-name .
```

means you must enter the keywords **CREATE TABLE** or **create table**.

abc Substitute a value for any term that appears in lowercase italic letters. In the previous example, you should substitute a value for *table-name*.

() Enter parentheses as shown. They are part of the syntax of a statement and are not special symbols.

[] Do not enter brackets as part of a statement; they surround any part of a statement that is optional. The statement

```
CREATE [UNIQUE] INDEX
```

indicates that you may enter either **create index** or **create unique index**.

| The vertical bar indicates a choice among several options. For example,

```
[ONE | TWO [THREE] | FOUR]
```

means that you can enter either **one** or **two** or **four** and that, if you enter **two**, you may also enter **three**. (Since the choices in this example are surrounded by square brackets, you can also omit them entirely.)

{ } When you must choose one of several options, the options are enclosed in braces and are separated by vertical bars.

{ONE | TWO | THREE}

means that you must enter one or two or three and that you may not enter more than one of them.

ABC When one of several options is the default, it appears underlined.

[CHOCOLATE | VANILLA | STRAWBERRY]

means that you may select any of the three options, but that if you do not enter one of them, VANILLA is the default.

... Enter additional items like those preceding the ellipsis, if you want. The ellipsis indicates that the immediately preceding item may be repeated indefinitely. For example,

statement

...

means that series of statements can follow the one that is listed.

Related Reading

Users who have had no prior experience with database management may want to refer to an introductory text like C. J. Date's *Database: A Primer* (Addison-Wesley Publishing, 1983).

Readers desiring more technical information may want to consult *An Introduction to Database Systems, Volume I* (Addison-Wesley Publishing, 1981) and *An Introduction to Database Systems, Volume II* (Addison-Wesley Publishing, 1983), also by C. J. Date.

This manual assumes you are familiar with your computer's operating system. Readers with little operating system experience may want to look at their operating system manual or a good introductory text before starting to learn about **INFORMIX-SQL**.

Suggested texts on UNIX are *A Practical Guide to the UNIX System* by M. Sobell (Benjamin/Cummings Publishing, 1984), *A Practical Guide to UNIX System V* by M. Sobell (Benjamin/Cummings Publishing, 1985), and *UNIX for People* by Birns, Brown, and Muster (Prentice-Hall, 1985).

The documentation that comes with all DOS systems is a useful source of information about that operating system.

Preparing to Use **INFORMIX-SQL**

This section describes the steps you must follow before running **INFORMIX-SQL**, summarizes the conventions for using your terminal with the program, and explains how to use the demonstration database provided as part of **INFORMIX-SQL**. It also explains how **INFORMIX-SQL** uses operating system file and directory permissions to control access to databases, and the ways in which operating system constraints may affect the performance of the program.

This section assumes that **INFORMIX-SQL** is installed on your computer according to the installation instructions in the letter that comes with the software package.

Setting Up the Operating Environment

Several environment variables affect how **INFORMIX-SQL** acts on your system. These variables are not identical across DOS and UNIX systems. Each system is addressed separately in the following two sections.

Setting Environment Variables on UNIX Systems

Before you can use **INFORMIX-SQL**, you must make the following changes to your computing environment:

- Set the **INFORMIXDIR** environment variable so that **INFORMIX-SQL** can locate its programs.
- Set the **TERM** environment variable so that **INFORMIX-SQL** recognizes the kind of terminal you are using.
- Set the **TERMCAP** environment variable so **INFORMIX-SQL** can communicate with your terminal.
- Set the **PATH** environment variable so that the shell searches the correct bin for executable **INFORMIX-SQL** programs.

You may set these environment variables at the system prompt or in your **.profile** (Bourne shell) or **.login** (C shell) file. If you set these variables at the system prompt, you will have to assign them again the next time you log onto the system. If you set these variables in your **.profile** or **.login** file, UNIX assigns them automatically every time you log in.

The following set of instructions assumes that **INFORMIX-SQL** resides in the directory **/usr/informix**. If, for some reason, **INFORMIX-SQL** has been installed in another directory, substitute the new directory name for **/usr/informix**.

1. Log onto the system.
2. Set the **INFORMIXDIR** environment variable.

If you are using the Bourne shell, enter

```
INFORMIXDIR=/usr/informix
export INFORMIXDIR
```

If you are using the C shell, enter

```
setenv INFORMIXDIR /usr/informix
```

3. Set the **TERM** environment variable. To do this, you need to obtain the code for your terminal from the system administrator.

If you are using the Bourne shell, enter

```
TERM=name
export TERM
```

where *name* is the code for the terminal you are using.

If you are using the C shell, enter

```
setenv TERM name
```

4. Set the **TERMCAP** environment variable. If the code for your terminal is listed in the **TERMCAP** file included with **INFORMIX-SQL**, follow the instructions below. Otherwise, consult your system administrator about the procedure for your system.

If you are using the Bourne shell, enter

```
TERMCAP=/usr/informix/etc/termcap
export TERMCAP
```

If you are using the C shell, enter

```
setenv TERMCAP /usr/informix/etc/termcap
```

5. Include the directory containing **INFORMIX-SQL** in your **PATH** environment variable.

If you are using the Bourne shell, enter

```
PATH=$PATH:/usr/informix/bin  
export PATH
```

If you are using the C shell, enter

```
setenv PATH ${PATH}:/usr/informix/bin
```

6. If you placed these environment settings in your **.login** or **.profile** file, you should log out and then log back in. (This allows the shell to read the changes you have made.) The new settings take effect each time you log in.

If you entered these environment settings from the system prompt, the new settings remain in effect until you log out. You will need to re-enter them every time you log onto the system.

In addition to the environment variables listed above, you may assign other environment variables by following the instructions in Appendix B.

Setting Environment Variables on DOS Systems

Before you can use **INFORMIX-SQL**, you must make the following changes to your computing environment:

- Use your system editor to set **FILES** to 20 (or more) and **BUFFERS** to 8 (or more) in your **CONFIG.SYS** file.
- Set the **INFORMIXDIR** environment variable so that **INFORMIX-SQL** can locate its programs.
- Use the **PATH** command to have DOS search **\informix\bin** for executable programs.

You may set the INFORMIXDIR environment variable and run the PATH command at the system prompt or include them in your AUTOEXEC.BAT file. If you execute them at the system prompt, you have to execute them each time you start the system. If you set these variables in your AUTOEXEC.BAT file, DOS assigns them automatically each time you start the system.

1. Turn your system on.
2. Use your system editor to assign the following values to FILES and BUFFERS in your CONFIG.SYS file:

```
FILES=20  
BUFFERS=8
```

These are suggested values only. If you anticipate establishing a large database or using **PERFORM** with multiple table forms, you may want to increase the FILES setting beyond 20. If your computer has 512K RAM or more, you may also want to increase the BUFFERS setting beyond 8. This increases performance, in most instances, by reducing the delay caused by excessive disk accesses.

After you change the **CONFIG.SYS** file, reset your computer.

Note: The following instructions assume that **INFORMIX-SQL** resides in the **\informix** directory. If, for some reason, **INFORMIX-SQL** has been installed in another directory, substitute the new directory name for **\informix**.

3. Set the INFORMIXDIR environment variable by entering

```
set INFORMIXDIR=\informix
```

4. Use the PATH command to have DOS search the **\informix\bin** directory for executable programs:

```
PATH \informix\bin[;dirname]
```


The brackets indicate an optional part of the command. Do not enter the brackets if you want to include other directories in the search path. Also, make sure you enter a semicolon between directory names.

5. If you placed the **INFORMIXDIR** and **PATH** command settings in your **AUTOEXEC.BAT** file, you should restart your computer. This allows the operating system to read the changes you have made.

If you did not include the **INFORMIXDIR** and **PATH** commands in your **AUTOEXEC.BAT** file, you will have to enter them again when you restart the system.

In addition to the environment variables listed above, you may assign other environment variables by following the instructions in Appendix B.

Using Your Terminal

Your computer keyboard looks something like a typewriter keyboard, except that the computer keyboard includes some special keys. You use some of these special keys to give instructions to **INFORMIX-SQL**. Before you begin using **INFORMIX-SQL**, locate the following keys on your keyboard:

RETURN	The RETURN key is sometimes labeled CR or NEWLINE , or is marked with a bent arrow. It is located on the right side of the keyboard.
CONTROL	The CONTROL key is often labeled CTRL or CNTRL and is located at the left of the keyboard. In this manual it is called CTRL .
LEFT ARROW	The LEFT ARROW key moves the cursor back one position on the screen. If your terminal does not have a LEFT ARROW key, use the CTRL-H key.

DOWN ARROW	The DOWN ARROW key moves the cursor down one line. If your terminal does not have a DOWN ARROW key, use the CTRL-J key.
UP ARROW	The UP ARROW key moves the cursor up one line. If your terminal does not have an UP ARROW key, use the CTRL-K key.
RIGHT ARROW	This key moves the cursor forward one position on the screen. If your terminal does not have a RIGHT ARROW key, use the CTRL-L key.
DELETE	<p>The DELETE key is sometimes labeled RUBOUT, CANCEL, or DELETE, and is referred to as DEL.</p> <p>On most UNIX systems the DEL key is used to cancel or abort a program, or leave a current menu and return to the menu one level above. It is referred to in this guide as the <i>program interrupt</i> key.</p>
CTRL-C	On DOS systems, and some UNIX systems, the CTRL-C key is used to cancel or abort a program, or leave a current menu and return to the menu one level above. It is referred to in this manual as the <i>program interrupt</i> key.
BACKSPACE	The BACKSPACE key may be labeled with a left-pointing arrow, and is usually located at the top right of the keyboard.
ESCAPE	The ESCAPE key is sometimes labeled ESC .
SPACEBAR	The SPACEBAR is generally unlabeled.

In addition, DOS systems have ten function keys marked **F1** to **F10**. The function keys are usually placed at the left of the keyboard.

The Demonstration Database

The **INFORMIX-SQL** package includes a demonstration database called **stores** that contains information about a fictitious wholesale sporting-goods distributor. The sample forms, reports, and command files that make up a demonstration application are also included. You can use this demonstration material to follow the examples in the manual.

For ease of reading, the **stores** database and the sample forms, reports, and command files that comprise the demonstration application are referred to as the “demonstration database.”

Creating the Demonstration Database

As you learn about **INFORMIX-SQL**, you will want to experiment with the demonstration database. To do this, you need to make a copy of this material. To make a copy of the demonstration database, select the directory where you want to store the material (often your home directory) and make it your current working directory. Then, from the operating system command line, enter

```
isqldemo
```

The **isqldemo** program creates a subdirectory called **stores.dbs** in your current directory and places the **stores** database files there. It also copies all the demonstration report, form, and command files into your current directory.

If you list the contents of your current directory, you will see filenames similar to these:

c_index.sql	ord1.ace	ex4.sql
c_custom.sql	ord1.arc	ex5.sql
c_items.sql	ord2.ace	ex6.sql
c_orders.sql	ord2.arc	ex7.sql
c_manuf.sql	ord3.ace	ex8.sql
c_stock.sql	ord3.arc	ex9.sql
c_stores.sql	maill.ace	ex10.sql
customer.per	mail1.arc	ex11.sql
customer.frm	mail2.ace	ex12.sql
orderform.per	mail2.arc	ex13.sql
orderform.frm	mail3.ace	ex14.sql
sample.per	mail3.arc	ex15.sql
sample.frm	ex1.sql	ex16.sql
clist1.ace	ex2.sql	ex17.sql
clist1.arc	ex3.sql	ex18.sql
clist2.ace		ex19.sql
clist2.arc		stores.dbs

Additional forms, reports, and command files have been included that are not part of the demonstration application. These provide further opportunities for practice once you have become familiar with the demonstration database.

Restoring the Demonstration Database

As you work with your copy of the demonstration database, you may make changes in such a way that the illustrations in this manual no longer reflect what you actually see on your screen. This can happen if you enter a great deal of new information into the **stores** database, delete the information that came with the database, or significantly alter the structure of the tables, forms, reports, or command files.

You can restore the demonstration database to an original condition (the one upon which the examples are based) by recreating the database with the `isqldemo` command.

You may want to make a fresh copy of the demonstration database each time you start a new chapter.

If you installed your **INFORMIX-SQL** software according to the instructions provided in your installation letter, the files that

make up the demonstration database are protected so that you cannot make any changes to the original copy.

Restricting Database Access on Multi-User Systems

INFORMIX-SQL includes a set of statements used to ensure database security and data integrity. You can control database and table privileges independently, allowing only selected individuals to access certain tables (or parts of tables) and excluding other users from the database entirely.

The **RDSQL** statements used for these purposes are described in Chapter 7 of the *INFORMIX-SQL User Guide* and Chapter 2 of the *INFORMIX-SQL Reference Manual*.

For **INFORMIX-SQL** to access a database, the current user must have execute (search) permission for each directory in the database pathname. Check with your system administrator for more information about UNIX file and directory permissions.

Chapter 1

Introducing INFORMIX-SQL

Chapter 1 Table of Contents

Chapter Overview	5
What Is INFORMIX-SQL?	6
What Is a Database?	7
How INFORMIX-SQL Stores Data	8
A Database Is a Collection of Tables.....	8
Tables Are Made Up of Rows and Columns.....	9
Retrieving Data from the Database.....	11
Searching for Rows	11
Searching for Columns	12
Searching for Columns within Rows.....	12
Connecting Tables through Column Joins.....	13
Getting Started with INFORMIX-SQL.....	16
Accessing INFORMIX-SQL on DOS Systems.....	16
Loading the Database Agent	17
Entering INFORMIX-SQL.....	17
The INFORMIX-SQL Main Menu	18
The Current Option.....	19
The Message Line.....	19
Moving the Highlight	20
Selecting an Option	20
What the Main Menu Options Mean	20
Using the HELP Menu	23
The INFORMIX-SQL Facilities.....	24
Creating Tables with the Schema Editor.....	24
The Screen Form Features	26
The Report Writing Facilities	29
The RDSQL Structured Query Language.....	32
The User-menu System.....	34
A Complete Database Management System.....	35
Chapter Summary.....	36

Chapter Overview

This chapter introduces **INFORMIX-SQL** and considers important database management concepts. The topics discussed include

- What is **INFORMIX-SQL**
- What is a database
- What are tables, rows, and columns
- What is a relational database management system
- What is a screen form
- What is a report
- What is a database query

To make the best use of **INFORMIX-SQL**, you must understand the basic concepts in this chapter. The section “What Is a Database?” is particularly helpful if you have had limited experience with computers or database management systems.

What Is INFORMIX-SQL?

INFORMIX-SQL is a computer-based record-keeping system. As a *database management system*, **INFORMIX-SQL** consists of useful programs or modules that perform data management tasks. **INFORMIX-SQL** can substantially reduce the amount of time required to organize, store, and retrieve information. It can summarize, group, and format information in a variety of helpful ways.

You can use **INFORMIX-SQL** to organize and store data, then view it in whatever format is useful to you. For example, you can store all the names, addresses, birthdays, and phone numbers of your friends, then use **INFORMIX-SQL** to search through the database for all the Hunts, everyone with a birthday in March, all the Sutherlands on Cambridge Avenue with birthdays in May, and so on.

The next section describes how **INFORMIX-SQL** stores and retrieves information. Included is a discussion of key database management concepts.

What Is a Database?

A *database* is a collection of information or *data*. The telephone directory is a database; its data is the names, addresses, and telephone numbers of local residents and businesses. Other kinds of databases include

- Mailing lists
- Inventory lists
- Customer lists

They are called databases because they consist of information that is useful to a particular organization or information that is used for a specific purpose.

The telephone directory is a good example of a database. Here is a sample:

Last Name	First Name	Address	Telephone
Hunt	Eloise	15 Emerson	415-987-6543
Hunt	Frank	1102 University	415-123-4567
Hunt	Hubert	95A Sacramento	415-254-1107
Hunt	Inez	99 California Ave	415-786-1111
Hunt	Jason	60 Oakdale	415-454-9087

The telephone directory is organized alphabetically by last name. If you want to call Jason Hunt, you look in the Last Name column under *H*. As long as you know the last name of the person you want to call, this scheme works just fine.

But what about finding phone numbers for everyone who lives on California Avenue? Or the phone number for someone whose last name you do not know? Since the telephone directory is not organized by street addresses or first names, it would take a long time to find these numbers.

This is where **INFORMIX-SQL** can help.

How INFORMIX-SQL Stores Data

INFORMIX-SQL stores information in tables. A table is a collection of rows and columns. Tables, rows, and columns are the building blocks that make up an INFORMIX-SQL database. This section explains these basic elements and how INFORMIX-SQL uses them.

A Database Is a Collection of Tables

A database is made up of *tables*. A table is a collection of information organized in *rows* and *columns*. A database contains at least one table and can contain as many tables as you need.

Each table in a database normally contains a different kind of information. For example, you would probably maintain separate tables for the products that you sell, the orders that you take, and the customers that you serve. You need to store different information for products, orders, and customers; that is why you create separate tables. At any time you can add a new table or delete a table you no longer need.

The INFORMIX-SQL demonstration database contains information about a fictitious wholesale sporting goods store. This **stores** database contains the following five tables:

- The **customer** table contains information about the retail sporting goods stores that purchase equipment from the wholesaler.
- The **orders** table contains information about recent purchases made by each retail sporting goods store.
- The **items** table contains information about the individual items contained in each order placed by the retail sporting goods stores.
- The **stock** table contains information about the items carried in stock by the wholesaler.

- The **manufact** table contains information about the various manufacturers who supply the wholesaler with products.

Some database management systems refer to tables as *files*.

Tables Are Made Up of Rows and Columns

INFORMIX-SQL tables consist of rows and columns, just like the tables you see everyday in books, newspapers, and on the evening news.

Each row contains all the data about one of the objects the table describes. For example, in the **stores** database

- Each row in the **customer** table contains all the information about one retail customer.
- Each row in the **orders** table contains all the information about one purchase made by a retail sporting goods store.
- Each row in the **items** table contains all the information about one item in an order placed by a retail sporting goods store.
- Each row in the **stock** table contains all the information about one of the wholesaler's products.
- Each row in the **manufact** table contains all the information about one manufacturer.

When you create a new table, it contains no rows. It consists only of the column names, and contains no data. When you begin to store data in a table, you usually add one row at a time.

Each table includes one or more columns. A column contains a particular type of information, such as a last name, order number, or zip code. When you create a database table, you assign a name to each column. These *column names* are similar to the headings that appear at the top of each column in a

printed table. You and **INFORMIX-SQL** use the column names to refer to the columns.

Some database management systems refer to rows as *records* and to columns as *fields*.

In the demonstration database, each row in the **customer** table contains 10 columns. The names of the columns and the information they contain are as follows:

This Column	Contains This Information
customer__num	Customer number
fname	Representative's first name
lname	Representative's last name
company	Name of store
address1	Store address, first line
address2	Store address, second line
city	City
state	State
zipcode	Zip code
phone	Phone number

Figure 1-1. Columns in the **customer** Table

The figure that follows shows some of the data from the **customer** table, organized into rows and columns.

Data in the <i>customer</i> Table				
customer__num	fname	lname	name	address1
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court
102	Carole	Sadler	Sports Sport	785 Geary St
103	Philip	Currie	Phil's Sports	654 Poplar
104	Anthony	Higgins	Play Ball!	East Shoping Cntr.
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive

Retrieving Data from the Database

Once you store some data in a database, you can ask **INFORMIX-SQL** questions about the data. This process is called *querying* the database. Querying means searching for information that is stored in the database.

You can query an **INFORMIX-SQL** database in three different ways. You can

- Search for a particular row or group of rows
- Search for a particular column or group of columns
- Combine both of these techniques to search for specific columns in specific rows

Additionally, you can use a technique called *joining* to retrieve data from more than one table at a time. The ability to retrieve information from multiple tables is one of the many powerful facilities of **INFORMIX-SQL**.

Searching for Rows

When you search for a row or group of rows in a database, **INFORMIX-SQL** displays all the information from each row. For example, you may want to

- Show all the information for the customer store named *All Sports Supplies*
- Show information for customers who have stores located in Redwood City
- Show information for customers who have stores located within the 94020 to 94040 zip code areas.

Each of these queries asks **INFORMIX-SQL** to display *all* the information in one or more rows. Some database management systems call this type of query *selection*.

Searching for Columns

You can also search for one or more columns. For example, you might want to list customer names and phone numbers, without addresses and zip codes. Alternately, you may want to

- Show the names of all stores and the cities in which they are located
- Show all store names, addresses, and phone numbers
- Show the zip codes of all stores

Each of these queries looks for *all* values in a particular column or group of columns. Some database management systems call this type of query a *projection*.

Searching for Columns within Rows

More frequently, you combine two techniques to find the data you want. For example, you can

- Show the name and phone number of each customer in the city of San Francisco. (**INFORMIX-SQL** finds the name and phone number columns from the rows for customers located in San Francisco.)
- Show the names of customers with stores in the 408 telephone area code. (**INFORMIX-SQL** finds the name columns from the rows for customers with telephone numbers beginning with the 408 area code.)
- Show the name and address of the store that has been assigned customer number 101. (**INFORMIX-SQL** finds the name and address columns from the row of the store assigned customer number 101.)

Each of these queries requests some, but not all, of the information stored in the **customer** table.

Connecting Tables through Column Joins

INFORMIX-SQL is a *relational* database management system. Unlike a file management system, a relational database management system lets you create direct *relationships* between tables when you query a database.

With **INFORMIX-SQL**, you do not need to duplicate the same information in different tables. Instead, you use a technique called *joining* to connect data from different tables. Joining lets you look at data stored in several tables as if it were all part of a single table.

The **stores** database provides a good example of the use of joins. You may periodically want to print a list showing each customer's name, address, and phone number, along with a list of each customer's recent orders. You could do this by storing complete customer information, along with order information, in the **orders** table. Doing so, however, would make maintaining the database more time-consuming and prone to errors.

The **customer** table contains information about the name, location, and phone number of each sporting goods customer. If you also stored this information in the **orders** table, you now have the name, location, and phone number stored in two separate tables. This means you need to update both the **customer** and **orders** tables whenever a customer changes address or phone number. Aside from being inconvenient, having to enter the same data in two tables creates greater opportunity for data-entry errors.

Duplicate table information is unnecessary with **INFORMIX-SQL**. You can obtain the list of recent customer orders more easily by joining tables.

Tables can be joined if one column in each table contains the same (or similar) data. This column is called the *join column*, and you use it to create a temporary link—or join—between the tables.

For example, the **customer** and **orders** tables both contain a **customer_num** (customer number) column. You can use this column as a join column. In the **customer** table, this column shows the unique customer number assigned to each sporting goods retailer. In the **orders** table, it shows which customer placed the order described in the row. By joining the two **customer_num** columns, you can obtain the desired list of customer name, address, and phone number (retrieved from the **customer** table) and orders (retrieved from the **orders** table) without duplicating information in the two tables. This joining process is illustrated in Figure 1-2.

			customer table (detail)
customer_num	fname	lname	
101	Ludwig	Pauli	
102	Carole	Sadler	
103	Philip	Currie	
104	Anthony	Higgins	

			orders table (detail)
order_num	order_date	customer_num	
1001	01/20/1986	104	
1002	06/01/1986	101	
1003	10/12/1986	104	
1004	04/12/1986	106	

Figure 1-2. Tables Joined by the **customer_num** Column

The row in the **customer** table with information about Anthony Higgins contains the number 104 in the **customer_num** column. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. According to Figure 1-2, customer 104 (Anthony Higgins) placed two orders, since his customer number appears in two rows in the **orders** table.

The ability to join data from different tables is one of the most important features of a truly *relational* database management system. Instead of storing the identical data in several tables, a relational database management system lets you retrieve data from several tables at once and display it as if it were stored in a *single* table.

Joining lets you rearrange your view of a database whenever you want. This flexibility lets you create new relationships between tables without redesigning your database. You can easily expand the scope of a database by adding new tables that join existing tables.

The five tables of the **stores** database are linked together by potential join columns, as shown in the next figure.

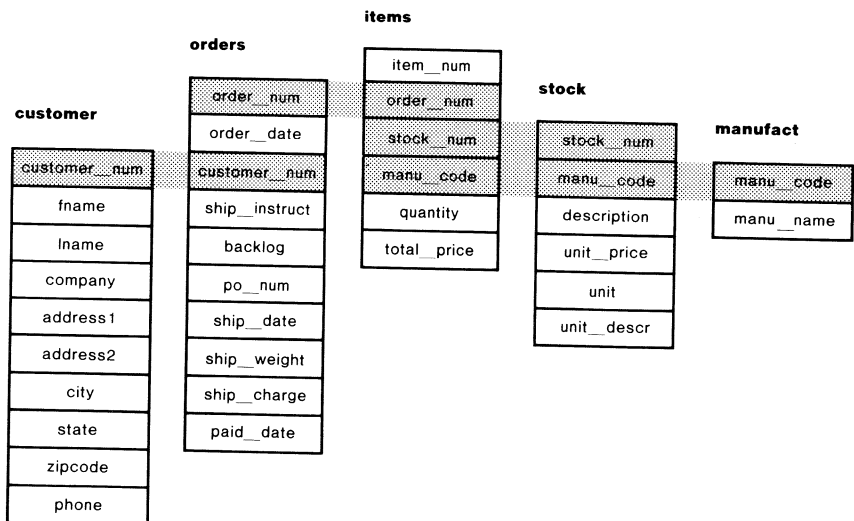


Figure 1-3. Join Relationships in the **stores** Database (potential join columns shaded)

Appendix A, “The *stores* Database,” includes a section discussing the join columns that link the tables in the demonstration database.

Getting Started with INFORMIX-SQL

To use **INFORMIX-SQL**, you need a database with which to work. The examples in this book are based on the **stores** demonstration database. You can follow the examples if you create a copy of the **stores** database, as explained in the section, “Creating the Demonstration Database,” in the Preface.

To start working with **INFORMIX-SQL**, be sure that your computer is up and running, and that the operating system prompt appears on your screen.

On UNIX systems, enter

```
isql
```

This operation displays the **INFORMIX-SQL** Main Menu as described in the section “The **INFORMIX-SQL** Main Menu.”

DOS system users must perform the additional steps outlined in the following section.

Accessing INFORMIX-SQL on DOS Systems

To access **INFORMIX-SQL**, on DOS systems you must perform two steps:

1. Load the *database agent*. The database agent is the **INFORMIX-SQL** program that performs the data access and file-manipulation tasks necessary to complete your database management operations.
2. Enter **INFORMIX-SQL**.

Loading the Database Agent

To load the database agent, enter

```
startsql
```

You need to execute this command only once per session. The database agent remains in memory until you execute the command

```
exit
```

from the command line. Executing this command removes the database agent from memory. Because the database agent requires space in memory, you should remove it when you have completed all the desired **INFORMIX-SQL** operations. This frees memory for running other programs on your system.

If you attempt to load the database agent when it is already installed in memory, **INFORMIX-SQL** issues the message **STARTSQL** has already been executed.

Entering INFORMIX-SQL

After you have loaded the database agent, enter

```
isql
```

If you try to run `isql` without first loading the database agent, **INFORMIX-SQL** issues the message **Please run STARTSQL**.

If you want to check to see if the database agent has been loaded, enter

```
set
```

This gives you a list of your environment variables. If the database agent has been loaded, the variable **SQLCADDR** is listed. The following is an example of **SQLCADDR** as it appears in a list of environment variables. The numbers represent the address in memory where the database agent is loaded.

```
SQLCADDR=333744a8
```

The INFORMIX-SQL Main Menu

The Informix logo is displayed, followed by the **INFORMIX-SQL** Main Menu.

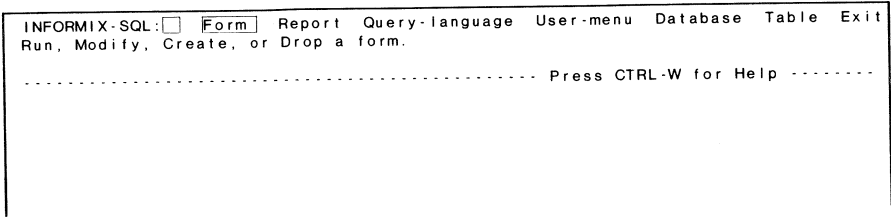


Figure 1-4. The INFORMIX-SQL Main Menu

Note: If **INFORMIX-SQL** does not display the Main Menu and instead displays

Unknown terminal type.

then it does not recognize the type of terminal you are using. Ask your system administrator for assistance, or consult the section, “Preparing to Use **INFORMIX-SQL**,” in the Preface. (**INFORMIX-SQL** may not be properly installed.)

The **INFORMIX-SQL** Main Menu displays a series of options across the top of the screen. These options represent the major **INFORMIX-SQL** functions.

The Current Option

One option is always marked as the *current option*. It is displayed in reverse video (dark letters on a light background) if your terminal can display reverse video, or surrounded by angle brackets < > if it cannot.

The screen illustrations in this manual show the reverse video as a box that surrounds the highlighted part of the screen. In the preceding figure, **Form** is the current option.

The Message Line

A message appears on the line immediately beneath the list of options. The message describes the current option. In the previous figure, the message

Run, Modify, Create, or Drop a form.

describes the activities available to you when you select the **Form** option on the INFORMIX-SQL Main Menu. You can

- *Run* an existing form. (Use a screen form to enter or retrieve data.)
- *Modify* an existing form. (Alter the appearance or behavior of the screen form.)
- *Create* a new screen form.
- *Drop* (delete) an existing form.

Moving the Highlight

You move the highlight by pressing the **SPACEBAR** or the **ARROW** keys. As you move the highlight, notice that the message displayed on the Message Line beneath the options changes. This message always describes the highlighted option. If you move the highlight over the **Report** option, you see the following message:

Run, Modify, Create, or Drop a report.

If you cannot remember what an option does, just highlight it and read the message.

Selecting an Option

Once you highlight the option you want, you can select it by pressing the **RETURN** key. Pressing **RETURN** tells **INFORMIX-SQL** you want to work with the option selected. For example, if you want to select, create, or drop a database, move the highlight to **Database** and press **RETURN**.

You can also select an option by typing the first letter of its name. Typing the first letter selects the option right away; you do not have to move the highlight or press **RETURN**. For example, no matter where the highlight appears on the Main Menu, you can always select the **Exit** option by pressing the **e** key.

What the Main Menu Options Mean

The options on the Main Menu give you access to all **INFORMIX-SQL** functions. Select an option on the Main Menu, and **INFORMIX-SQL** displays a new menu of options relating to your choice. For example, if you select the **Table** option, you see another menu that lets you specify whether you want to create, alter, display information about, or drop a database table.

Here is a summary of the INFORMIX-SQL Main Menu options.

Form	Displays the FORM Menu. Use this menu to work with screen forms. Screen forms are described in Chapter 3 through Chapter 6.
Report	Displays the REPORT Menu. Use this menu to work with reports. A report selects data from one or more tables in a database and prints it in the format you specify. See Chapter 8, “Creating and Printing Reports,” for details.
Query-language	Displays the RDSQL Menu. Use this menu to work with the RDSQL query language. You use this language to search a database for information, add information to a database, and change the structure of a database. See Chapter 7, “Using RDSQL,” for details.
User-menu	Displays the USER-MENU Menu. Use this menu to work with custom screen menus. These menus are created through the procedures described in Chapter 6, “User-menu,” in the <i>INFORMIX-SQL Reference Manual</i> .
Database	Displays the DATABASE Menu. Use this menu to select, create, or drop a database.
Table	Displays the TABLE Menu. Use this menu to create or alter a table.
Exit	Exits INFORMIX-SQL and returns to the operating system.

The following figure illustrates the **INFORMIX-SQL** menu structure.

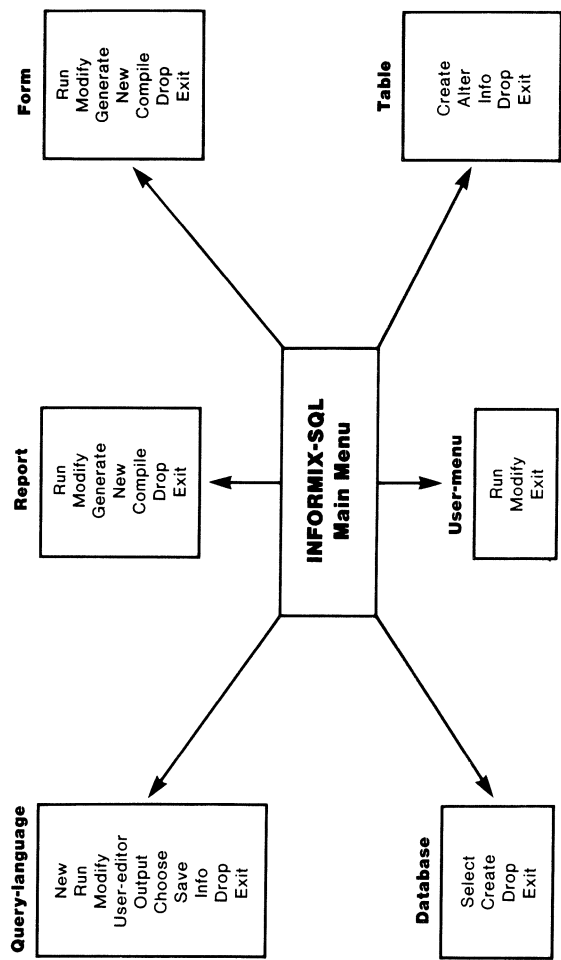


Figure 1-5. The **INFORMIX-SQL** Menu Structure

Using the *HELP* Menu

Every **INFORMIX-SQL** screen has a **HELP** Menu associated with it. The **HELP** Menu contains information about your current options and suggests appropriate actions. Whenever you want information about what to do next, press the **CTRL-W** key, and **INFORMIX-SQL** displays the **HELP** Menu. On DOS systems, you may also use the **F1** key to access the **HELP** Menu.

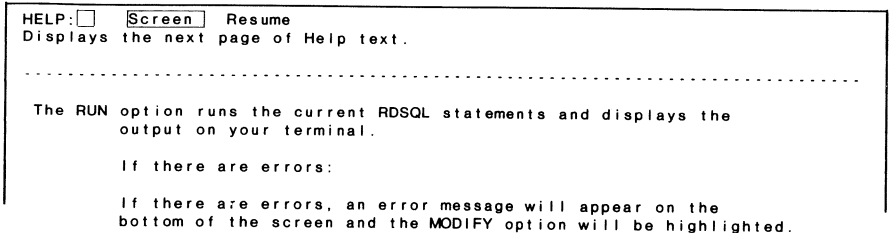


Figure 1-6. The **HELP** Menu with Sample Text

You can read as much or as little text as you need. Selecting the **Screen** option on the **HELP** Menu displays the next page of the **HELP** text; selecting the **Resume** option lets you continue your work.

You use the **HELP** Menu in the same way you use other **INFORMIX-SQL** menus. Press the **SPACEBAR** to move the highlight to the option you want, and press **RETURN**. You can also enter the first letter of an option.

The INFORMIX-SQL Facilities

INFORMIX-SQL provides a complete set of database management facilities that allow you to perform a wide range of tasks. With INFORMIX-SQL you can

- Create and modify tables using the menus provided with the *schema editor*
- Enter and retrieve database information using *screen forms*
- Sort, combine, format, and display data with *reports*
- Enter, modify, and retrieve database information using the *query language*
- Access INFORMIX-SQL through special purpose, custom *menus*

Each of these features is demonstrated in this section.

Creating Tables with the Schema Editor

A database is made up of one or more tables. The INFORMIX-SQL schema editor provides menus that guide you through the steps necessary to create each table in your database.

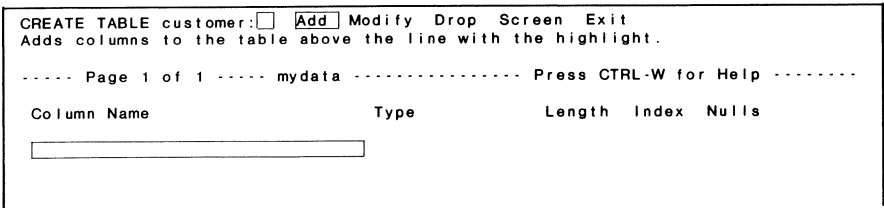


Figure 1-7. The CREATE TABLE Menu in the Schema Editor

The CREATE TABLE Menu lets you create the *schema* for a table on the bottom portion of the screen. A schema is essentially a blueprint for a table. Just as a blueprint of a building provides instructions for constructing the building, a table schema defines the structure of a database table. The **dbschema** utility allows you to reproduce a table schema. See Appendix D of the *INFORMIX-SQL Reference Manual* for a detailed description of this utility.

The finished table schema for the **customer** table (included in the demonstration database) appears as follows:

CREATE TABLE customer : ☐ ☒ Add Modify Drop Screen Exit
Adds columns to the table above the line with the highlight.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name	Type	Length	Index	Nulls
customer_num	Serial	101	Unique	No
fname	Char	15		Yes
lname	Char	15		Yes
company	Char	20		Yes
address1	Char	20		Yes
address2	Char	20		Yes
city	Char	15		Yes
state	Char	2		Yes
zipcode	Char	5	Dups	Yes
phone	Char	18		Yes

Figure 1-8. The Schema for the **customer** Table

The Screen Form Features

Once you create a table, you can store data in it. The easiest way to enter the data you want to store is to use a *screen form*. Much like a printed form, a screen form contains blank spaces that you can fill in. The blank spaces on a screen form are called *fields*.

Included with the demonstration database application material is a screen form named **customer** that you can use to enter data into the **customer** table. Here is what the **customer** form looks like:

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table. ** 1: customer table**

CUSTOMERS

Customer Number: []

Company : []

First Name: [] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

Figure 1-9. The **customer** Form

You add data to a table by entering it into the fields on a form. Each field normally corresponds to a column in one of your database tables. The data you enter into a field is stored in the corresponding column.

You can also use a screen form to query the database and retrieve information. For example, to find the rows for all customers with stores in Redwood City, you would complete the **customer** form as follows:

```
QUERY:  ESCAPE queries.  DEL discards query.  ARROW keys move cursor.
Searches the active database table.          ** 1: customer table**
```

CUSTOMERS

Customer Number: []

Company : []

First Name: [] Last Name: []

Address : []

City : [Redwood City] State : [] Zip : []

Telephone : []

Figure 1-10. The Query Screen (UNIX Systems)

This screen instructs **INFORMIX-SQL** to search for rows in which the **city** column contains “Redwood City.”

Here is what the **customer** form looks like when you execute the query:

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table. ** 1: customer table**

CUSTOMERS

Customer Number: [104]

Company: [Play Ball]

First Name: [Anthony] Last Name: [Higgins]

Address: [East Shopping Cntr.]
[422 Bay Road]

City: [Redwood City] State: [CA] Zip: [94026]

Telephone: [415-368-1100]

5 row(s) found

Figure 1-11. A Sample Screen Form Query

The message at the bottom of the screen indicates that **INFORMIX-SQL** found five customers with locations in Redwood City. The data about the first customer appears on the screen.

The Report Writing Facilities

A report consists of database information, arranged and formatted according to your instructions, and displayed on the screen, printed on paper, or stored in a file for future use. You can select all or some of the information stored in a database and display it in a report.

The **INFORMIX-SQL** report writer retrieves data from a database, organizes and formats it according to your instructions, adds the headings or column titles you specify, performs calculations on numeric columns, then displays the report.

You can combine information from several tables in a database to create whatever kind of report you need. Some of the reports **INFORMIX-SQL** can produce are

- Mailing labels
- Form letters
- Payroll summaries
- Medical histories
- Accounting records
- Inventory lists
- Telephone directories
- Project schedules
- Sales and marketing reports
- Payroll checks

The demonstration database application materials include a variety of reports illustrating the uses of the **INFORMIX-SQL** report writing facilities. A sample report follows.

CUSTOMER LIST

September 30, 1986

Last Name	Company	City
Vector Lawson	Los Altos Sports Runners & Others	Los Altos Los Altos
Beatty Grant	Sportstown Gold Medal Sports	Menlo Park Menlo Park
Watson Parmelee	Watson & Son Olympic City	Mountain View Mountain View
Baxter	Blue Ribbon Sports	Oakland
Currie Ream	Phil's Sports Athletic Supplies	Palo Alto Palo Alto
Higgins Quinn Jaeger Albertson Sipes	Play Ball! Quinn's Sports AA Athletics Sporting Place Kids Korner	Redwood City Redwood City Redwood City Redwood City Redwood City
Sadler	Sports Spot	San Francisco
Pauli Miller Keyes	All Sports Supplies Sport Stuff Sports Center	Sunnyvale Sunnyvale Sunnyvale

Total Number of Customers: 18

Figure 1-12. A Sample Report

This report lists the last name, company, and city of all customers. It groups customers by city, and calculates the total number of customers.

You can also prepare interactive reports with **INFORMIX-SQL**. The next report asks the user for beginning and ending dates and prepares a listing of daily orders. It then automatically computes and prints subtotals and totals.

DAILY ORDER REPORT				
FROM: 01/01/86		RUN DATE: 08/20/86		
TO: 03/30/86		RUN TIME: 16:20:54		
ORDER DATE	COMPANY	NAME	NUMBER	AMOUNT
01/20/1986	Play Ball!	Anthony Higgins	1001	\$250.00
Total amount ordered for the day:				\$250.00
02/14/1986	Sports Center	Frances Keyes	1009	\$450.00
Total amount ordered for the day:				\$450.00
03/23/1986	Play Ball!	Anthony Higgins	1011	\$99.00
Total amount ordered for the day:				\$99.00
03/25/1986	Kids Korner	Arnold Sipes	1007	\$1,696.00
Total amount ordered for the day:				\$1,696.00
Total amount ordered for all days:				\$2,495.00

Figure 1-13. A Second Sample Report

The RDSQL Structured Query Language

RDSQL is an English-like, interactive query language that you can use when working with databases. **RDSQL** is an enhanced version of SQL (Structured Query Language), the industry-standard query language developed by IBM. With **RDSQL** you can perform a wide variety of database management tasks, including

- Create and drop tables
- Enter and delete data
- Query a database
- Rename tables and columns

To use **RDSQL**, you enter one or more **RDSQL statements**. A statement is simply an instruction that tells **RDSQL** what you want to do. For example, to create a table, you use the **CREATE TABLE** statement; to query a database, you use the **SELECT** statement.

Here is the statement that was used to create the **stock** table in the demonstration database:

```
create table stock

(   stock_num      smallint,
    manu_code      char(3),
    description     char(15),
    unit_price      money(6),
    unit            char(4),
    unit_descr      char(15)
);
```

This statement tells **RDSQL** to create a table named **stock** and specifies the name of each column and the type of data (numbers, words, money, or dates) each column will contain. Column names appear on the left; data types are indicated on the right.

The next example illustrates how database queries are easily achieved with RDSQL. In this example all stock items relating to the sport of baseball are selected from the database, and a five percent increase in their unit price is calculated.

```
select stock_num, stock.manu_code,
       manu_name, description, unit_price,
       unit_price * 1.05 newprice
from stock, manufact
where stock.manu_code
= manufact.manu_code and
description matches "baseball*"
order by stock.manu_code
```

The next figure shows the results of this query.

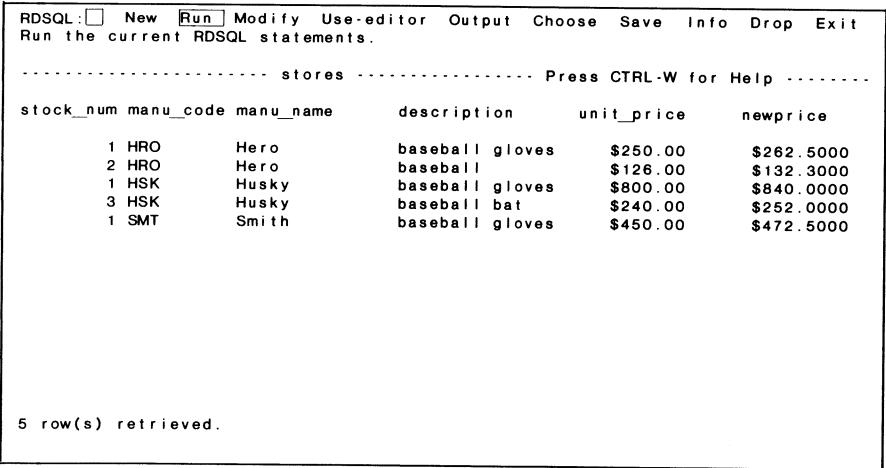


Figure 1-14. A Sample RDSQL Query

The User-menu System

The **INFORMIX-SQL User-menu** facility is used to create and run custom menus. The options on a user-menu can call

- Submenus on the custom menu system
- **INFORMIX-SQL** programs (reports, screen forms, and **RDSQL** queries, for example)
- Other programs or sets of programs in your software library
- Operating system utilities

The main level of the user-menu included with the demonstration database application materials appears as the next figure.

```
WEST COAST WHOLESALERS, INC.  
  
1. FORMS  
2. REPORTS  
3. QUERIES  
4. TABLE DEFINITIONS  
5. UTILITIES  
  
Use space bar, arrow keys, or type number to make selection.  
Enter 'e' to return to previous menu or exit.  
Enter carriage return to execute selection: 1
```

Figure 1-15. The Main Menu of the **stores** User-menu

You can create custom menus that allow individuals with limited experience to use sophisticated **INFORMIX-SQL** database management applications.

Complete Database Management System

The **INFORMIX-SQL** facilities equip you to perform all your database management tasks. You can easily add new tables to a database—and modify existing tables—with the schema editor. You can add, update, and delete data with screen forms. You can use the **RDSQL** query language to enter and retrieve database information, and modify the structure of the database. You can display database information in a variety of formats with reports. You can create custom menus that provide all users with easy access to **INFORMIX-SQL** programs.

Chapter Summary

- A database is a collection of information or data organized into one or more tables.
- Each table consists of data organized into rows and columns.
- Each column contains a particular type of information about each row in the table.
- Each row contains a complete set of data about one of the objects described in the table.
- Recalling information from a database is called querying the database.
- You can query for a row or group of rows, for a column or group of columns, or for certain columns within certain rows.
- **INFORMIX-SQL** is a relational database management system. This means you have the flexibility to create relationships across tables when you query a database.
- Instead of duplicating information in several tables, a technique called joining lets you look at data stored in several tables as if it were all part of a single table.
- The **INFORMIX-SQL Main Menu** gives you access to everything you can do with **INFORMIX-SQL**.
- A **HELP Menu** is available to describe each menu option.

Chapter 2

Creating a Database

Chapter 2 Table of Contents

Chapter Overview	5
Creating the Demonstration Database	5
Two Ways to Create Databases and Tables	6
Using the Main Menu to Create Databases and Tables	6
Using RDSQL to Create Databases and Tables.....	6
Accessing INFORMIX-SQL on DOS Systems.....	7
Loading the Database Agent	7
Entering INFORMIX-SQL.....	8
Accessing INFORMIX-SQL on UNIX Systems	9
Using the Main Menu to Create a Database.....	10
Naming the New Database	11
Guidelines for Selecting Database Names.....	11
The Current Database.....	12
System Files that Make Up a Database.....	13
Selecting a Different Current Database	14
Leaving a Current Menu or Screen	15
Updating an Existing INFORMIX-SQL Database.....	15
Using the Main Menu to Create a Table.....	16
Naming the New Table	17
Using the CREATE TABLE Menu.....	18
An Overview of Building a Table.....	18
Defining Each Column in the Table	19
Correcting Mistakes.....	19
Naming the Columns.....	20
Assigning Data Types.....	21
CHAR Columns.....	22
Numeric Columns	22
SERIAL Columns	24
DATE Columns.....	24
MONEY Columns.....	25
Assigning a Data Type to the Column	25

Creating Indexes	2
When to Index Columns.....	2
NULL Values.....	2
Adding the Remaining Columns to the Table	3
<i>fname</i> column	3
<i>lname</i> column	3
<i>company</i> column	3
<i>address1</i> column	3
<i>address2</i> column	3
<i>city</i> column	3
<i>state</i> column	3
<i>zipcode</i> column.....	3
<i>phone</i> column	3
Exiting the Schema Editor	3
System Files.....	3
Changing the Table Specifications	3
Chapter Summary.....	3

Chapter Overview

This chapter explains how to set up an **INFORMIX-SQL** database and describes how to create both a table and an index. The following topics are discussed:

- How to create a database
- How to identify the current database
- How to create a table
- What are data types
- How to modify the structure of a table
- When to create an index
- How to create an index
- What are NULL values

If you intend only to use databases that someone else has created, you can skim this chapter. Otherwise, read it carefully, for it contains important information that you will need to set up your own **INFORMIX-SQL** databases.

Creating the Demonstration Database

If you have not already done so, create the demonstration database as described earlier in the section “Creating the Demonstration Database,” in the Preface.

Two Ways to Create Databases and Tables

INFORMIX-SQL provides two methods for creating databases and tables: selecting the **Database** and **Table** options on the **INFORMIX-SQL Main Menu** or using the **RDSQL** query language.

Using the Main Menu to Create Databases and Tables

As described in this chapter, the easiest way to create a database or table is to select the **Database** and **Table** options on the **INFORMIX-SQL Main Menu**. Menus are provided at each step, and you are prompted to enter each piece of necessary information. **HELP** Menus are also available for added assistance.

This chapter provides a solid base for exploring the **RDSQL** query language in Chapter 7, “Using **RDSQL**.”

Using RDSQL to Create Databases and Tables

RDSQL, the **INFORMIX-SQL** *query language*, is based on **SQL** (Structured Query Language), the industry-standard query language developed by IBM. The **RDSQL** query language provides a flexible and efficient method for performing a variety of functions. This section introduces **RDSQL**, while Chapter 7 describes in further detail the use of **RDSQL** statements for creating a database and table.

RDSQL is a set of English-like statements you can use to perform the following functions:

- Create and drop tables and indexes
- Enter and delete data
- Query a database
- Select a different current database
- Display information about one or more tables
- Rename tables and columns
- Check and repair tables
- Grant and revoke database and table privileges

For complete information about the **RDSQL** commands and statements, see Chapter 2 of the *INFORMIX-SQL Reference Manual*.

Before you can store data using **INFORMIX-SQL**, you must create a database and one or more tables. This chapter explains how these tasks are performed.

Accessing **INFORMIX-SQL** on DOS Systems

Before working with **INFORMIX-SQL**, be sure your computer is running properly and that the operating system prompt appears on the screen.

Loading the Database Agent

To access **INFORMIX-SQL**, you must first load the *database agent*. To load the database agent, enter

```
startsql
```

You need to execute this command only once per session.

The database agent remains in memory until you execute the command

```
exit
```

from the command line. Executing this command removes the database agent from memory. Because the database agent requires space in memory, you should remove it when you have completed all the desired **INFORMIX-SQL** operations. This frees memory for running other programs on your system.

If you attempt to load the database agent when it is already installed in memory, **INFORMIX-SQL** issues the message **STARTSQL** has already been executed.

Entering INFORMIX-SQL

After you have loaded the database agent, enter

`isql`

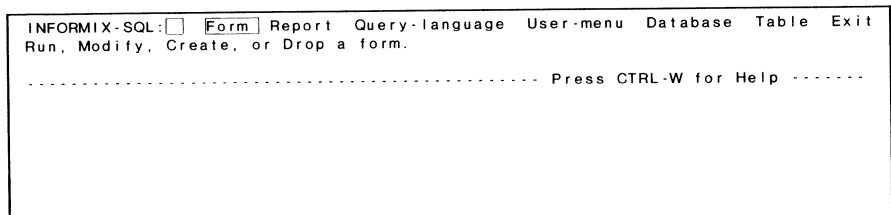
If you try to run `isql` without first loading the database agent, **INFORMIX-SQL** issues the message **Please run STARTSQL**.

If you want to check to see if the database agent has been loaded, enter

`set`

This gives you a list of your environment variables. If the database agent has been loaded, the variable `SQLCADDR` is listed.

When **INFORMIX-SQL** is ready to be used, the Informix logo appears, followed by the **INFORMIX-SQL Main Menu**.



```
INFORMIX-SQL: ☐ ☒ Form Report Query-language User-menu Database Table Exit
Run, Modify, Create, or Drop a form.
----- Press CTRL-W for Help -----
```

Accessing INFORMIX-SQL on UNIX Systems

Before working with **INFORMIX-SQL**, be sure that your computer is running properly and that the operating system prompt appears on the screen. Then enter

`isql`

When **INFORMIX-SQL** is ready to be used, the Informix logo appears, followed by the **INFORMIX-SQL Main Menu**.

```
INFORMIX-SQL: ☐ ☒ Form ☐ Report ☐ Query-language ☐ User-menu ☐ Database ☐ Table ☐ Exit
Run, Modify, Create, or Drop a form.
```

```
----- Press CTRL-W for Help -----
```

Using the Main Menu to Create a Database

To create a database, select the **Database** option on the Main Menu.

```
INFORMIX-SQL: ☐ Form Report Query-language User-menu Database Table Exit
Select, Create, or Drop a database.

----- Press CTRL-W for Help -----
```

INFORMIX-SQL then displays the DATABASE Menu.

```
DATABASE: ☐ Select Create Drop Exit
Select a database to work with.

----- Press CTRL-W for Help -----
```

Four options appear on the DATABASE Menu.

Select	Selects an existing database
Create	Creates a new database
Drop	Drops an existing database
Exit	Exits to the INFORMIX-SQL Main Menu

To create a database, select the **Create** option.

```
DATABASE: ☐ Select Create Drop Exit
Create a new database.

----- Press CTRL-W for Help -----
```

Naming the New Database

The CREATE DATABASE Screen appears and prompts you to name the new database.

```
CREATE DATABASE >>  
Enter the name you want to assign to the new database, then press Return.  
..... Press CTRL-W for Help .....
```

Type the new database name (this chapter uses **mydata**) and press **RETURN**. You can assign any name you want, as long as you follow the guidelines below. If you accidentally enter the name of an existing database, **INFORMIX-SQL** displays the message

330: Cannot create database.

Use another name.

Guidelines for Selecting Database Names

When choosing a database name, you are limited in the number and types of characters you can use, and **INFORMIX-SQL** reserved words are not allowed.

Number of Characters. The name you assign to the database may be from one to eight (on DOS systems) or ten (on UNIX systems) characters long. If you store more than one database in a directory, each database name must be unique.

Characters You Can Use. The database name must begin with a letter. The rest of the name may consist of any combination of letters, numbers, and underscores (_). You may use uppercase and lowercase letters, but **INFORMIX-SQL** makes no distinction between them. Therefore, **tahiti**, **Tahiti**, and **TAHITI** all refer to the same database.

Reserved Words. Certain *reserved words* have special meanings to **INFORMIX-SQL**, and you should not use them to name a database, table, or column. These are words like *create*, *database*, *file*, *group*, *master*, *output*, *prompt*, and *table*. A complete list of the reserved words appears in Appendix C.

The Current Database

The database you are currently working with is called the *current database*. **INFORMIX-SQL** displays the name of the current database in the middle of the dashed line that separates the top and bottom of the screen. When you create a new database, it automatically becomes the current database.

The DATABASE Menu now displays the name of the current database.

DATABASE: <input type="text"/> <input type="button" value="Select"/> <input type="button" value="Create"/> <input type="button" value="Drop"/> <input type="button" value="Exit"/>				
Select a database to work with.				
..... mydata Press CTRL-W for Help				

System Files that Make Up a Database

When you create a database, **INFORMIX-SQL** creates a directory to store the information about the database, and places the new directory in the current directory. **INFORMIX-SQL** names the directory with the database name you selected and appends the extension **.dbs** (on UNIX systems) or **.DBS** (on DOS systems). Example **INFORMIX-SQL** database directories are

On UNIX Systems:

```
/usr/norma/products.dbs  
/usr/andrea/schedule.dbs
```

On DOS Systems:

```
\NORMA\PRODUCTS.DBS  
\ANDREA\SCHEDULE.DBS
```

The database directory initially contains 18 files. The files are called *system catalogs*. **INFORMIX-SQL** uses the system catalogs to keep track of the tables, columns, indexes, views, synonyms, and permissions in each database. The system catalogs are described in detail in Appendix K of the *INFORMIX-SQL Reference Manual*.

Selecting a Different Current Database

The **Database** option on the Main Menu allows you to select a different current database. When using the DATABASE Menu, your screen looks something like this:

```
DATABASE: ☐ ☒ Select Create Drop Exit
Select a database to work with.

----- mydata ----- Press CTRL-W for Help -----
```

To select a different database, move the highlight to the **Select** option and press the **RETURN** key, or type **s**. The CHOOSE DATABASE Screen appears.

```
CHOOSE DATABASE >> ☐
Choose a database with the Arrow Keys, or enter a name, then press Return.

----- mydata ----- Press CTRL-W for Help -----

mydata
stores
```

You select a database by typing its name and pressing **RETURN** or by using the **ARROW** keys to move the highlight to the name of the database you want. Then press **RETURN**, and the name under the highlight becomes your current database. For example, to select the **stores** database in the menu above, type or highlight **stores**, and then press **RETURN**.

To leave the CHOOSE DATABASE Screen without selecting a new database, press **DEL** (on most UNIX systems) or **CTRL-C** (on all DOS systems and some UNIX systems).

Exit the DATABASE Menu by typing **e**.

Leaving a Current Menu or Screen

You always have the option of leaving the current menu or screen and moving up a level in the menu hierarchy. On most menus, this is accomplished by selecting the **Exit** menu option.

As you saw with the CHOOSE DATABASE Screen, there are times when you do not have an **Exit** option—most screens do not contain this option, or you may be working with a part of **INFORMIX-SQL** that does not use menus. You can always exit a screen *without making any choices or changes* by pressing **DEL** (in most UNIX systems) or **CTRL-C** (in DOS and some UNIX systems). Pressing this *program interrupt* key moves you out of the current screen and up a level in the menu hierarchy.

Updating an Existing INFORMIX-SQL Database

Databases created with pre-2.00.00 versions of **INFORMIX-SQL** must be updated to be used with the new options available on the current version of **INFORMIX-SQL**. Appendix I of the *INFORMIX-SQL Reference Manual* describes how to update a database.

Databases created with **INFORMIX** must be converted to be used with the current version of **INFORMIX-SQL**. Appendix H of the *INFORMIX-SQL Reference Manual* describes how to convert a database.

Using the Main Menu to Create a Table

Once you create a database, the next step is to create the tables you want included in the database.

INFORMIX-SQL does not limit the number of tables in a database; the limit is determined by the amount of disk space available on your computer.

Any tables created by **INFORMIX-SQL** are placed in the current database. Before you create a table, be sure that the current database is the one in which you want the table to exist. The name of the current database is displayed on the third line from the top of the screen.

The following examples indicate you are working with the **mydata** database; this is reflected in the status line that appears at the top of the screen. If **mydata** is not your current database, use the options available with the DATABASE Menu and select (or create) **mydata**.

To begin creating a table, select the **Table** option on the **INFORMIX-SQL** Main Menu. The **TABLE** Menu appears next.

TABLE:	<input type="checkbox"/>	<input checked="" type="checkbox"/> Create	<input type="checkbox"/> Alter	<input type="checkbox"/> Info	<input type="checkbox"/> Drop	<input type="checkbox"/> Exit
Create a new table.						
..... mydata Press CTRL-W for Help						

The TABLE Menu displays six options:

Create	Creates a new table
Alter	Alters the structure of an existing table
Info	Gives information about the structure of a table
Drop	Drops a table from the database
Exit	Exits to the INFORMIX-SQL Main Menu

To create a table, select the **Create** option of the TABLE Menu.

TABLE: <input type="checkbox"/> <input checked="" type="checkbox"/> Create <input type="checkbox"/> Alter <input type="checkbox"/> Info <input type="checkbox"/> Drop <input type="checkbox"/> Exit
Create a new table.
----- mydata ----- Press CTRL-W for Help -----

Naming the New Table

The CREATE TABLE Screen appears and prompts you to name the new table.

CREATE TABLE >> <input type="checkbox"/>
Enter the table name you wish to create with the schema editor.
----- mydata ----- Press CTRL-W for Help -----

You must follow the same guidelines that apply to database names (see the section “Guidelines for Selecting Database Names,” earlier in this chapter) except that table names can be up to 18 characters long.

Enter a table name (this chapter uses the name **clients**) and press **RETURN**.

INFORMIX-SQL next displays the **CREATE TABLE** Menu.

Using the CREATE TABLE Menu

The **CREATE TABLE** Menu guides you through the steps necessary to create a table.

CREATE TABLE clients: <input type="checkbox"/> Add <input type="checkbox"/> Modify <input type="checkbox"/> Drop <input type="checkbox"/> Screen <input type="checkbox"/> Exit				
Adds columns to the table above the line with the highlight.				
----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----				
Column Name	Type	Length	Index	Nulls
<input type="text"/>				

The **CREATE TABLE** Menu displays five options:

- | | |
|---------------|---|
| Add | Adds a new column to the table |
| Modify | Modifies the structure of an existing column |
| Drop | Drops an existing column from the table |
| Screen | Scrolls down the screen and displays new text |
| Exit | Exits to the TABLE Menu |

An Overview of Building a Table

The **CREATE TABLE** Menu displays the *schema* for the table on the bottom portion of the screen. A schema is essentially a blueprint for a table. Just as a blueprint of a building provides instructions for how the building is constructed, a table schema defines the columns in a database table.

Each horizontal line in the schema represents one column, with the name of the column at the left. Each of the five headings in the schema presents some information about the column, such as the length of the column and the type of data it will store.

The **CREATE TABLE** Menu works with the *schema editor* to design the table schema. Whenever you select the **Create** or **Alter** options on the **TABLE** Menu to create or alter a table schema, you use the schema editor.

Defining Each Column in the Table

In creating a table schema, you define each column in the table, one at a time. As you define each column, **INFORMIX-SQL** prompts you for the information it needs. Most of the questions **INFORMIX-SQL** asks are in the form of menus so you can select the appropriate response quickly and easily. Some of the questions, such as the name of the column, require you to enter something other than a menu selection. As you answer these questions, **INFORMIX-SQL** places your answers in the schema. You move from left to right across the screen as you define each column, and from top to bottom as you define additional columns.

Correcting Mistakes

If you make a mistake entering information about a column, you can back up and correct it as long as you are still defining the same column (still working on the same line). Use the **LEFT ARROW** and **RIGHT ARROW** keys to move the highlight back and forth under the headings. If you notice a mistake in one table column after you have moved on to another, refer to the section “Changing the Table Specifications,” later in this chapter.

After you have finished building a schema for the **clients** table, the screen will appear as follows:

```

CREATE TABLE clients : ☐ Add ☐ Modify ☐ Drop ☐ Screen ☐ Exit
Adds columns to the table above the line with the highlight.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name                Type                Length    Index Nulls
customer_num                Serial              101       Unique No
fname                       Char                15        Yes
lname                       Char                15        Yes
company                     Char                20        Yes
address1                     Char                20        Yes
address2                     Char                20        Yes
city                        Char                15        Yes
state                       Char                 2        Yes
zipcode                     Char                 5        Dups Yes
phone                       Char                18        Yes

```

The **clients** table created in this chapter is identical to the **customer** table included in the **stores** demonstration database

Note: The screen displays that follow are on a UNIX system. The screen on a DOS system is identical except for the program interrupt key: the **CTRL-C** key is used on all DOS systems and some UNIX systems, while the **DEL** key is used on the majority of UNIX systems.

Naming the Columns

Select the **Add** option on the **CREATE TABLE** Menu, and **INFORMIX-SQL** displays the **ADD NAME** Screen and prompts you for the column name.

```

ADD NAME >> ☐
Enter column name. RETURN adds it. DEL returns to CREATE TABLE menu.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name                Type                Length    Index Nulls

```

In selecting column names, you must follow the same guidelines that apply to table names (see the section “Naming the New Table,” earlier in this chapter). You must use a different name for each column within a single table so that **INFORMIX-SQL** can identify each column uniquely.

No DOS restrictions apply to column names, so you do not have to make the first eight characters unique (but the entire column name must still be unique).

Enter `customer__num` for the column name and press the **RETURN** key.

Assigning Data Types

The **ADD TYPE** Menu appears next.

ADD TYPE clients : <input type="checkbox"/> Char <input checked="" type="checkbox"/> Numeric <input type="checkbox"/> Serial <input type="checkbox"/> Date <input type="checkbox"/> Money			
Permits any combination of letters, numbers, and punctuation.			
----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----			
Column Name	Type	Length	Index Nulls
customer__num	<input type="text"/>		

You can store many different types of data in a table: dates, zip codes, names, part numbers, part descriptions, salaries, costs—the list is endless. You need to choose the *data type* that best suits each type of data. The data type indicates the kind of information you intend to store in each column.

Character Character columns store any combination of letters, numbers, and symbols.

Numeric There are six different numeric data types (including the Serial type described next). Each is designed to store a different kind of numeric data.

Serial Serial columns store sequential integers assigned by **INFORMIX-SQL**.

Date The date data type stores calendar dates.

Money Money columns store currency amounts.

A detailed explanation of data types appears in Chapter 9, “Database Structure and Integrity.” The next section provides a brief set of guidelines.

CHAR Columns

CHAR (short for CHARacter) columns store any combination of letters, numbers, and symbols. You normally use CHAR columns to store names, addresses, social security numbers, job titles, project names, and other information that combines letters, numbers, and symbols.

You must specify the length of each CHAR column. This length is the total number of characters you want to reserve for the name, address, job title, or whatever data you wish to store in the column.

Numeric Columns

There are six data types for numbers. Each is designed for a different kind of number. You cannot store characters or symbols in any numeric column, although you can use plus signs (+) and minus signs (–) to show whether numbers are positive or negative.

To indicate a NUMERIC data type, enter numeric when prompted by the ADD TYPE Menu. When you specify a NUMERIC data type, **INFORMIX-SQL** displays the ADD NUMERIC Menu and asks for the exact data type.

A summary of the numeric data types follows. The Serial data type is described separately in the next section.

DECIMAL. DECIMAL data types are numbers with definable precision and scale.

To use a DECIMAL data type, enter **decimal** when the ADD NUMERIC Menu appears. You can then stipulate the precision and scale, if desired. The total number of digits you want the decimal to hold is the *precision*, and the number of digits after the decimal point is the *scale*. Specifying the precision and scale is optional.

For example, the phrase (6,2) indicates that the column stores six-digit numbers, with four digits before the decimal point and two after the decimal point.

SMALLINT and INTEGER Columns. SMALLINT and INTEGER columns store whole numbers—numbers that have no fractional portion. SMALLINT columns store whole numbers from $-32,767$ to $+32,767$. INTEGER columns store whole numbers from $-2,147,483,647$ to $+2,147,483,647$.

To use these data types, enter **smallint** or **integer** when the ADD NUMERIC Menu appears.

SMALLFLOAT and FLOAT Columns. SMALLFLOAT and FLOAT columns store binary floating-point numbers. SMALLFLOAT columns contain single-precision, floating-point numbers with approximately seven significant digits. FLOAT columns contain double-precision, floating-point numbers with approximately fourteen significant digits. FLOAT columns do not store larger numbers; they store numbers with greater precision.

To use these data types, enter **smallfloat** or **float** when the ADD NUMERIC Menu appears.

SERIAL Columns

Serial numbers ensure that a unique number is assigned to each row of the table. You do not need to enter data in a SERIAL column. Each time you add a new row to a table, **INFORMIX-SQL** automatically assigns the next number, in sequence, to the SERIAL column. Normally, the starting number is one, but you can select any number greater than zero as your starting number. The highest serial number **INFORMIX-SQL** can assign is 2,147,483,647.

Once **INFORMIX-SQL** assigns a SERIAL number, you cannot change it. (If you were allowed to change the number, **INFORMIX-SQL** could not guarantee uniqueness.) Each table in a database may contain only one SERIAL column.

To create a SERIAL column, enter **serial** under the **Type** heading in the **CREATE TABLE** schema.

When you specify a SERIAL data type, **INFORMIX-SQL** displays the **ADD STARTING NUMBER** Screen and asks you for the starting number. You can select any starting number greater than zero. Do not enter a comma in the number you select. For example, you would enter 62000 to start the numbering at 62,000.

DATE Columns

Use the DATE data type for columns in which you want to store calendar dates. A DATE column holds a date in the form *mm/dd/yyyy* where *mm* is the month (1-12), *dd* is the day of the month (1-31), and *yyyy* is the year (0001-9999).

DATE value	Meaning
05/02/1985	May 2, 1985
09/08/1883	September 8, 1883
12/25/1900	December 25, 1900

To create a DATE column, enter **date** under the **Type** heading in the **CREATE TABLE** schema.

MONEY Columns

Money columns store currency amounts.

To create a MONEY column, enter money under the Type heading from the ADD TYPE Menu. INFORMIX-SQL displays the ADD LENGTH Screen and prompts you for the length (precision) of the MONEY column. You first enter the length (total number of digits that the MONEY column will accommodate), and press the RETURN key. You are then presented with the ADD SCALE Screen. Enter the number of digits that should appear to the right of the decimal point and press the RETURN key.

Assigning a Data Type to the Column

With an understanding of data types, you are now ready to assign a type to the **customer_num** column.

The ADD TYPE Menu should still appear on the screen.

ADD TYPE clients : <input type="checkbox"/> Char <input checked="" type="checkbox"/> Numeric <input type="checkbox"/> Serial <input type="checkbox"/> Date <input type="checkbox"/> Money			
Permits any combination of letters, numbers, and punctuation.			
----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----			
Column Name	Type	Length	Index Nulls
customer_num	<input type="text"/>		

The **customer_num** column will contain the customer number for each store. Data type SERIAL is appropriate for this column because you want to ensure that a unique number is assigned to each customer.

Type s and INFORMIX-SQL displays the ADD STARTING NUMBER Screen.

ADD STARTING NUMBER >>

Enter the starting number. RETURN adds it.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name	Type	Length	Index	Nulls
customer_num	Serial	<input type="text" value="1"/>	Unique	No

INFORMIX-SQL displays the SERIAL data type under the Type heading of the schema and, because you are describing a SERIAL column, asks you for the number INFORMIX-SQL should begin with when assigning numbers. INFORMIX-SQL automatically starts assigning SERIAL numbers at the number 1. If you want an alternate starting number, indicate it at this time; otherwise, press RETURN to select the default.

Enter 101 to begin the numbering sequence at 101.

INFORMIX-SQL automatically inserts Unique under the Index heading and No under the Nulls heading. INFORMIX-SQL assumes you want to index a SERIAL column and, because it is type SERIAL, duplicate values should not exist and NULL values should not be allowed. (Indexes and NULL values are discussed in the next two sections.)

Assigning the Serial data type completes the definition of the **customer_num** column.

The highlight moves down a line and to the left, the ADD NAME Menu appears, and **INFORMIX-SQL** is ready for you to begin defining the next column.

ADD NAME >>

Enter column name. RETURN adds it. DEL returns to CREATE TABLE Menu.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name	Type	Length	Index Nulls
customer_num	Serial	101	Unique No

Creating Indexes

Creating indexes for the columns in a table helps **INFORMIX-SQL** find information more quickly. However, just as you can find information in a book without consulting the index, **INFORMIX-SQL** can find information in a database even if you do not index any columns.

In fact, if there is only one table in your database, you probably will not need any indexes until you have entered several hundred rows of data into the table. Too many indexes has the undesired effect of slowing down the system. As your database increases in size and your database needs change, you can always add and delete indexes.

A detailed discussion of indexes and indexing strategies is contained in Chapter 9, “Database Structure and Integrity.” This section summarizes the material in that chapter.

When to Index Columns

It usually makes sense to index only columns that are included in queries or sorts. For example, if you generally sort personnel information alphabetically by last name, it makes sense to index the column that contains last names.

Indexing Guidelines. Indexing choices depend on how you use your database. In general, you do not need to create an index on a column until one or more of the following conditions exist:

- A table contains at least 200 rows.
- A column is used in a join operation.
- A column is regularly used in searches or sorts.

What You Can Index. You can create an index that applies to a single column or to several columns as a group.

Columns You Should Not Index. You cannot create an index for a column or group of columns whose total length exceeds 120 bytes. You should not create an index on a column that contains a large number of duplicate values. The section “Column and Row Lengths” in Chapter 9 includes more information about column length.

Preventing Duplicate Entries. **INFORMIX-SQL** normally allows you to enter the same value in different rows of an indexed column. In other words, even if you index a **last_name** column, you can still enter information about many people whose last name is Smith.

You may sometimes want to prevent users from entering duplicate information in an indexed column. If you create a column to store social security numbers, you would want to prevent duplicate entries. To prevent duplicate entries, enter unique from the **ADD INDEX** Menu.

Ascending and Descending Indexes. Unless you instruct otherwise, **INFORMIX-SQL** creates indexes in ascending order. You can also create descending indexes. You should create descending indexes only for fields you intend to regularly sort in descending order.

Clustered Indexes. Since both UNIX and DOS extract information from the disk in blocks, the more rows that are physically on the same block and are already in the same order as an index, the faster an indexed retrieval proceeds. You can cause the physical order in the table to be the same as the order in an index through *clustering*. See Chapter 2 of the *INFORMIX-SQL Reference Manual* for more information.

NULL Values

INFORMIX-SQL lets you identify columns in which NULL values are allowed. The purpose of NULL values in a database is to indicate when no value is assigned to a particular column in a particular row of a table. The reasons for not having assigned a value could include not knowing the correct value or that no value yet exists. The NULL may also indicate that no value is appropriate for a given column because of the values that were entered in other columns.

For example, consider entering data for a bank customer who is requesting a loan. If the customer is not employed, the **employer** column in the bank's **customer** table will not contain an entry for this person. This CHAR column has the value NULL. A **hire__date** column in the same table is meaningless in this case. There is no appropriate date to enter; the value is NULL.

Adding the Remaining Columns to the Table

To add the remaining columns to the **clients** table, repeat the steps used to enter the **customer__num** column into the table.

The following is a list of the questions that **INFORMIX-SQL** will ask about each of the database columns and an appropriate response for each question.

fname column

Column Name	fname
Type	char
Length	15
Index	no
Nulls	yes

lname column

Column Name	lname
Type	char
Length	15
Index	no
Nulls	yes

company column

Column Name	company
Type	char
Length	20
Index	no
Nulls	yes

***address1* column**

Column Name	address1
Type	char
Length	20
Index	no
Nulls	yes

***address2* column**

Column Name	address2
Type	char
Length	20
Index	no
Nulls	yes

***city* column**

Column Name	city
Type	char
Length	15
Index	no
Nulls	yes

***state* column**

Column Name	state
Type	char
Length	2
Index	no
Nulls	yes

zipcode column

Column Name	zipcode
Type	char
Length	5
Index	duplicates
Nulls	yes

phone column

Column Name	phone
Type	char
Length	18
Index	no
Nulls	yes

Exiting the Schema Editor

After you enter the final piece of information for the definition of the last column (the **phone** column) in the **clients** table, **INFORMIX-SQL** moves the highlight under the Column Name heading and displays the ADD NAME Screen. Since this is the last column, press **RETURN**, and **INFORMIX-SQL** exits the ADD NAME Screen and displays the CREATE TABLE Menu.

CREATE TABLE clients: ☐ ☒ Add ☐ Modify ☐ Drop ☐ Screen ☐ Exit

Adds columns to the table above the line with the highlight.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name	Type	Length	Index	Nulls
customer_num	Serial	101	Unique	No
fname	Char	15		Yes
lname	Char	15		Yes
company	Char	20		Yes
address1	Char	20		Yes
address2	Char	20		Yes
city	Char	15		Yes
state	Char	2		Yes
zipcode	Char	5	Dups	Yes
phone	Char	18		Yes

The schema for the **clients** table is now complete. Select the **Exit** option. **INFORMIX-SQL** displays the EXIT Menu.

EXIT clients: ☐ ☒ Build-new-table ☐ Discard-new-table

Builds a new table and returns to the Table Menu.

----- Page 1 of 1 ----- mydata ----- Press CTRL-W for Help -----

Column Name	Type	Length	Index	Nulls
customer_num	Serial	101	Unique	No
fname	Char	15		Yes
lname	Char	15		Yes
company	Char	20		Yes
address1	Char	20		Yes
address2	Char	20		Yes
city	Char	15		Yes
state	Char	2		Yes
zipcode	Char	5	Dups	Yes
phone	Char	18		Yes

The EXIT Menu displays two options:

Build-new-table Builds the table
Discard-new-table Discards the table instructions

Select the **Build-new-table** option.

System Files

For each table you create, **RDSQL** creates two system files. **INFORMIX-SQL** stores these files in the same directory as the system catalogs and names them with the first several characters of the name of the table, a unique number starting at 100, and the following extensions:

On UNIX Systems:

.dat	This file contains table data.
.idx	This file contains table indexes.

On DOS Systems:

.DAT	This file contains table data.
.IDX	This file contains table indexes.

When you use the operating system command to list the contents of the *mydata.dbs* directory, you see the names of these files along with the names of the system catalogs. Do not do anything with these files; **INFORMIX-SQL** uses them to keep track of the information in the table, and if you delete them, you also delete your table.

Changing the Table Specifications

Once you set up the schema for a table—even after you have put data into the table—**INFORMIX-SQL** allows you to change the way you store your data. Occasionally, making a change of this kind may also result in lost data. **INFORMIX-SQL** always advises you of this situation and gives you the option of not making the change. An example of a change that could cause a loss of data is changing a **CHAR** column from a length of 20 to 10. **INFORMIX-SQL** would drop (truncate) the last ten characters of this column in each row in the table.

The **TABLE** Menu includes an **Alter** option used to change a schema. Select the **Alter** option and **INFORMIX-SQL** prompts you for the name of the table you wish to modify. Once you enter the table name, **INFORMIX-SQL** displays the **ALTER TABLE** Menu and accesses the schema editor.

The **ALTER TABLE** Menu displays five options:

- | | |
|---------------|--|
| Add | Adds a column to the table |
| Modify | Modifies a column definition by changing one or more of the five schema headings (Column Name, Type, Length, Index, and Nulls) |
| Drop | Removes an entire column by erasing one line of the schema |
| Screen | Scrolls down the screen |
| Exit | Returns to the TABLE Menu |

The **Alter** option on the **TABLE** Menu is described in detail in the section “Changing the Structure of a Table” in Chapter 9, “Database Structure and Integrity.”

Chapter Summary

- The database you are working with is called the *current database*. The name of the current database always appears in the middle of the dashed line that separates the top and bottom of the screen.
- When you create a new database, it automatically becomes the current database.
- **INFORMIX-SQL** includes the **RDSQL** query language. **RDSQL** consists of English-like statements used specifically to work with a database.
- When creating a new table with the schema editor, you specify the name, data type, whether an index is established, and whether NULL values are allowed for each column.
- An index can apply to one or more columns in a single table. Assigning a **UNIQUE INDEX** to a column prevents users from entering duplicate values.
- You can use the schema editor to create a new table, add new columns to a table, modify a column, or drop a column from a table.

Chapter 3

Entering Data

Chapter 3 Table of Contents

Chapter Overview	5
What Is a Screen Form?	6
Screen Forms in the Demonstration Database	6
What Is PERFORM?	8
Including Data from Multiple Tables	8
Each Table Can Contribute to Several Forms	9
Using a Screen Form	9
Choosing the Form You Want	10
The PERFORM Screen	11
Information Lines	12
Screen Form	13
Status Lines	13
The PERFORM Menu Options	14
Adding Data to a Database	16
Entering Data in the Fields	18
Numeric Fields	18
Character Fields	19
DATE Fields	19
MONEY Fields	19
Confirming Your Entry	20
Data Checking in PERFORM	20
Special Functions	21
Positioning the Cursor	21
Field Editing	22
Storing the Row	24
Changing Existing Data	25
Removing a Row	26
Exiting PERFORM	27
Chapter Summary	28

Chapter Overview

This chapter explains how to enter and store data in an **INFORMIX-SQL** database. The topics discussed include

- What is a screen form
- What are fields
- What are field delimiters
- How to add a new row to a table
- How to change an existing row in a table
- How to delete a row from a table

This chapter covers important features that you should be familiar with before going on to Chapter 4, “Querying a Database.”

For complete information about entering data on a screen form, refer to Chapter 4, “The PERFORM Screen Transaction Processor,” in the *INFORMIX-SQL Reference Manual*.

What Is a Screen Form?

Once you create a table, you can store data in it. The easiest way to enter the data you want to store is to use a *screen form*. Much like a printed form, a screen form contains blank spaces that you can fill in. The blank spaces on a screen form are called *fields*.

You can create your own screen forms or use forms that someone else creates. Each screen form is assigned a name when the form is created. This chapter explains how to use an existing form. Chapter 6, “Creating Your Own Forms,” explains how to create screen forms.

Screen Forms in the Demonstration Database

The demonstration database includes a screen form named **customer** that you can use to enter data into the **customer** table. You will soon learn how to select this form. Here is what the **customer** form looks like when it appears on the screen:

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table. ** 1: customer table**

CUSTOMERS

Customer Number: []

Company : []

First Name: [] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

Figure 3-1. The customer Form

You add data to a table by entering it in the fields on a form. Each field normally corresponds to a column in one of your database tables. The data you enter in a field is stored in the corresponding column. The correspondences between fields and columns are specified when the form is created.

For example, the Company field on the **customer** form corresponds to the **company** column in the **customer** table. Whatever you enter in the Company field is stored in the **company** column in the **customer** table.

Fields presented on the screen are surrounded by brackets [] called *delimiters*. You can specify different delimiters when you create a form if you do not want to use the brackets. You normally include a descriptive label alongside each field to indicate what data is in the field. For example, the **customer** form contains labels like Customer Number and Name.

Horizontal and vertical lines can be included in your form. The **customer** form demonstrates the use of horizontal lines.

Fields can be displayed in reverse video. The Company field in the **customer** form appears in reverse video.

A form may be one or several pages long. If a form includes more data than can fit comfortably on a single screen, you can split the form over several screen “pages.”

What Is PERFORM?

When you use a screen form, you are actually using a part of **INFORMIX-SQL** called **PERFORM**.

You can access **PERFORM** from the **INFORMIX-SQL** Main Menu, and because it is a separate program, you can also call it up directly from the operating system. For details, refer to Appendix M of the *INFORMIX-SQL Reference Manual*.

Including Data from Multiple Tables

A single form can include data from several tables. The open-file limitation on your system limits the number of tables included in a form. The tables must all be part of the same database.

The demonstration database includes two forms that include data from several tables. The **orderform** form combines data from the **customer**, **orders**, **items**, and **manufact** tables. The **sample** form combines data from the **customer**, **orders**, **items**, **stock**, and **manufact** tables.

When you use a form that contains data from multiple tables, one table is always the *active table*. The active table is the table with which you are currently working. **PERFORM** stores the data you enter in the active table. If you delete data, **PERFORM** deletes it from the active table. You can easily determine which fields on the screen belong to the active table because they are surrounded by brackets (or the alternate delimiters you selected).

If a form includes data from only one table, that table is always the active table. You can read more about multiple-table forms in Chapter 5, "Using Multiple-Table Forms."

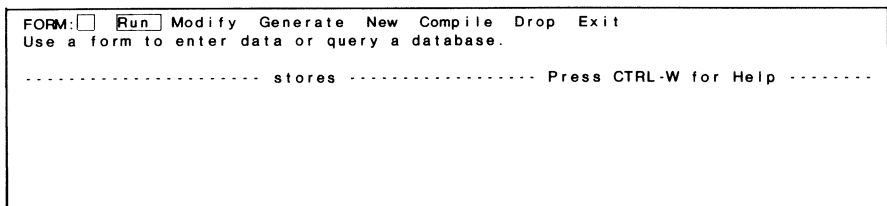
Each Table Can Contribute to Several Forms

There is no practical limit to the number of forms you can create for a single database. The various forms may contain different information or may display the same information in different ways. By creating multiple forms, you can combine information in whatever formats are most useful to you.

For example, the demonstration database includes one form for customer information (the **customer** form), a second form that combines customer, order, item, and manufacturer information (the **orderform** form), and a third form (the **sample** form) that uses information from all five tables in the **stores** database. The **customer** table contributes information to each form.

Using a Screen Form

You access **PERFORM** by selecting the **Form** option on the INFORMIX-SQL Main Menu. The FORM Menu then appears:



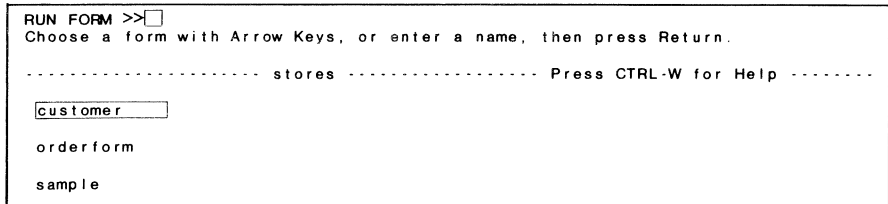
```
FORM: ☐ Run ☐ Modify ☐ Generate ☐ New ☐ Compile ☐ Drop ☐ Exit
Use a form to enter data or query a database.
..... stores ..... Press CTRL-W for Help .....
```

Figure 3-2. The FORM Menu

To use a screen form, select the **Run** option on the FORM Menu. **PERFORM** displays the RUN FORM Screen and a list of available forms.

Choosing the Form You Want

The *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual* include examples from the *customer*, *orderform*, and *sample* forms. An additional form has been included should you desire further practice once you have become familiar with these forms. When you select the **Run** option on the FORM Menu, you see the RUN FORM Screen.



```
RUN FORM >>☐
Choose a form with Arrow Keys, or enter a name, then press Return.

----- stores ----- Press CTRL-W for Help -----

customer
orderform
sample
```

Figure 3-3. The RUN FORM Screen

Choose the form you want by highlighting its name with the **ARROW** keys and then pressing **RETURN**. If you prefer, you can type the name of the form and then press the **RETURN** key.

This chapter uses the **customer** form and the **stores** database in its examples. To select the **customer** form, position the highlight on **customer** and press the **RETURN** key.

The PERFORM Screen

Once you select a form, **INFORMIX-SQL** pauses for a few moments as it locates the form and loads it into **PERFORM**. The **PERFORM** screen with the selected form then appears. The **customer** form is displayed in Figure 3-4.

The screenshot shows the PERFORM screen interface. At the top, there is a menu bar with options: **PERFORM:** Query Next Previous Add Update Remove Table Screen Below the menu bar, a status line reads: "Searches the active database table. ** 1: customer table**". The main area of the screen is titled "CUSTOMERS" and contains a form with the following fields: "Customer Number: []", "Company: []", "First Name: []", "Last Name: []", "Address: []", "City: []", "State: []", "Zip: []", and "Telephone: []". The form is enclosed in a rectangular border.

Figure 3-4. The **customer** Form

The **PERFORM** screen is divided into three sections: Information Lines, the Screen Form, and Status Lines.

Information Lines

The **PERFORM** menu appears on the top two lines of the screen (also called Information Lines). The first line displays a list of menu options; the second line describes the current option and indicates the number and name of the active database table. The next screen illustrates how the **PERFORM** menu initially appears.

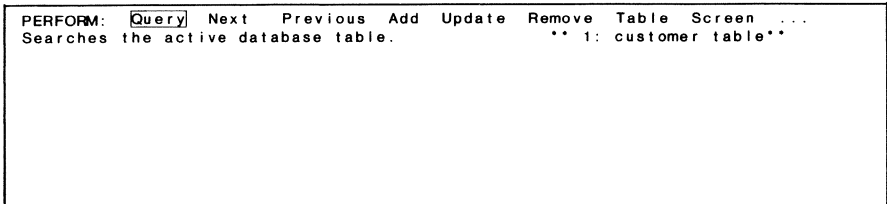
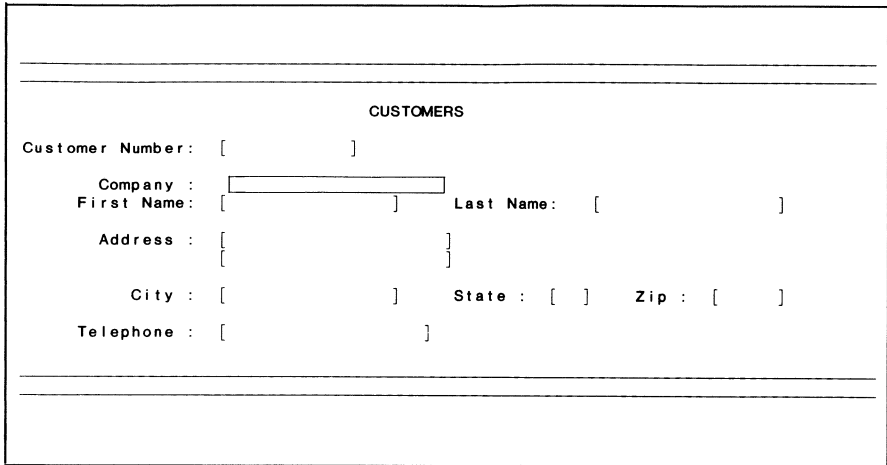


Figure 3-5. The PERFORM Menu

PERFORM assigns a number to each table contributing data to the screen form; if there are three tables, **PERFORM** numbers them 1, 2, and 3. When there is only one table involved in the screen form, it is given the number 1. In the sample figure, the **customer** table is both Table 1 and the active table.

Screen Form

The middle section of the screen displays the form you select. In this example, it displays the **customer** form.



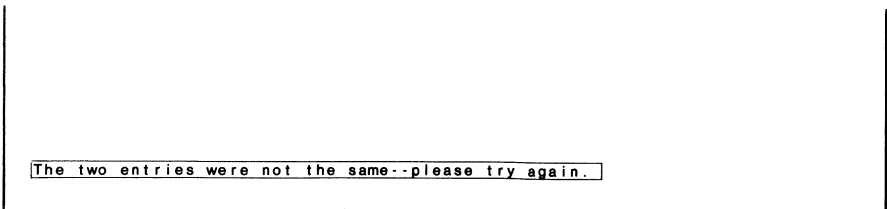
A screenshot of a screen form titled "CUSTOMERS". The form contains several input fields for customer information. The fields are arranged in a structured layout with labels and input boxes. The form is enclosed in a rectangular border with horizontal lines at the top and bottom.

CUSTOMERS			
Customer Number:	[]		
Company :	[]		
First Name:	[]	Last Name:	[]
Address :	[]		
City :	[]	State :	[] Zip :
Telephone :	[]		

Figure 3-6. The **customer** Screen Form (without the Information Lines)

Status Lines

PERFORM uses the last two lines of the screen to display error messages, as well as any messages generated by the form itself.



A screenshot of the bottom section of a screen, showing a status line with an error message. The message is displayed in a rectangular box at the bottom of the screen.

The two entries were not the same--please try again.

Figure 3-7. The Status Lines on a **PERFORM** Screen (Sample Error Message)

The PERFORM Menu Options

The **PERFORM** Menu is two pages long. The first page is illustrated below.

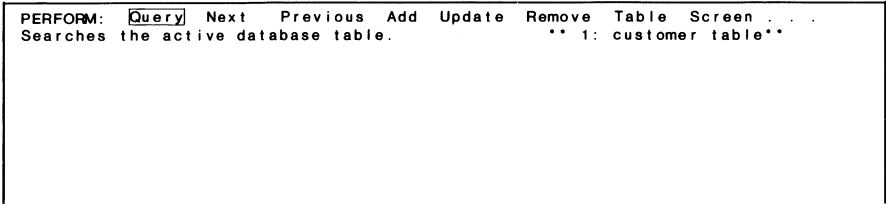


Figure 3-8. The First Page of the PERFORM Menu

Note: The number of options that appears on the first menu page depends upon the character capacity of your screen. The two page screens displayed here demonstrate a terminal or monitor with an 80 character screen. Terminals with a larger character capacity will show more options on the first menu page.

The ellipsis (...) at the end of the first line of the screen indicates that additional menu items are available on the second page of the menu. Move the highlight past the **Screen** option to display the second menu page. The second menu page is illustrated below.

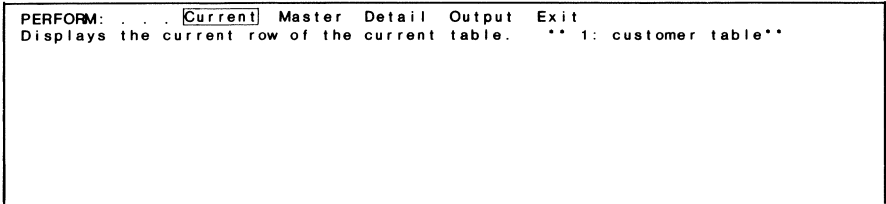


Figure 3-9. The Second Page of the PERFORM Menu

The ellipsis on the second page indicates that additional items are available on the previous menu page. Use the **SPACEBAR** or the **ARROW** keys to move the highlight onto the menu options. When you move the highlight past the first or last menu option on a page, the alternate menu page appears.

The **LEFT ARROW** moves the highlight to the left and, after the first item, moves the highlight to the last item on the alternate page. The **UP ARROW** and **DOWN ARROW** move the highlight to the first item on the alternate page. The **RIGHT ARROW** moves the highlight to the right and, after the last item, moves the highlight to the first item on the alternate page.

You select an option in one of two ways:

- Positioning the highlight on a menu option and pressing **RETURN**
- Typing the first letter of the option name, regardless of whether the option appears on the current menu page

The **Add**, **Update**, **Remove**, and **Exit** options are considered in detail in this chapter. The **Query**, **Next**, and **Previous** options are briefly mentioned in this chapter and are discussed in further detail in Chapter 4, “Querying a Database.” The **Screen** and **Output** options are also discussed in Chapter 4, and the remaining options are explained in Chapter 5, “Using Multiple-Table Forms.”

Adding Data to a Database

Use the **Add** option on the **PERFORM** Menu to create a new row in the active table. For example, you would use **Add** to create an entry in the **customer** table for a newly acquired customer.

Follow these steps to use the **Add** option:

1. Type **a** to select the **Add** option. **PERFORM** displays the **ADD** Menu and the cursor moves into the first field on the form.
2. Enter the data you want to add into the first field on the form (the **Company** field).
3. Press the **TAB** or **RETURN** key to shift the cursor to the next field.
4. Enter the data you want to add into the next field on the form (the **First Name** field).
5. Press the **TAB** or **RETURN** key to shift the cursor to the next field, and enter the appropriate data.
6. Repeat this process until you have entered the desired information into all fields in the form.

Note: the **State** field only accepts the values **ca**, **or**, **nv**, and **wa**.

7. Press **ESC** to add the new row data to the table. Press the program interrupt key to cancel the **Add** option without storing the row.

The next two screens display the information lines present upon selecting the **Add** option on the PERFORM Menu. The first screen shows the ADD Menu on most UNIX systems. The second screen shows the ADD Menu on DOS and some UNIX systems. Differences in the two screens are noted following the DOS screen.

```
ADD:  ESCAPE adds new data.  DEL discards it.  ARROW keys move cursor.  
Add new data to the active database table.      ** 1: customer table**
```

Figure 3-10. The ADD Menu (UNIX systems)

```
ADD:  ESCAPE adds new data.  CTRL-C discards it.  ARROW keys move cursor.  
Add new data to the active database table.      ** 1: customer table**
```

Figure 3-11. The ADD Menu (DOS systems)

The two screens appear identical except for the following change:

Interrupt The phrase DEL discards it. is replaced on DOS systems with CTRL-C discards it.

Entering Data in the Fields

What you can enter in a field depends on the data type of the column to which the field corresponds. There are four major types of fields:

- Numeric
- Character
- Date
- Money

Numeric Fields

Numeric fields correspond to SMALLINT, INTEGER, SMALLFLOAT, FLOAT, DECIMAL, or SERIAL columns. Complete information about data types appears in Chapter 9, “Database Structure and Integrity.”

You can enter only numbers in numeric fields. You must enter the correct type of number in each numeric field—integers in INTEGER fields, floating-point numbers in FLOAT fields, and so on.

PERFORM automatically assigns values to SERIAL fields when you add data to a database. You cannot enter numbers in SERIAL fields and, once **INFORMIX-SQL** assigns a serial number, you cannot change it.

As you enter data on the **customer** form, notice that the cursor will not move into the Customer Number field because the corresponding column in the **customer** table is a SERIAL field. **PERFORM** displays a serial number in the Customer Number field when you confirm the data you have entered by pressing the ESC key.

Character Fields

Character fields in **PERFORM** correspond to **CHAR** columns in a database table. You can enter letters, numbers, and symbols in character fields. You can enter as many characters as you can fit in the field.

A character field is normally the same length as the **CHAR** column to which it corresponds. However, if the field is longer than the column, you may occasionally enter more characters than the column can hold. If you do, **PERFORM** stores as many characters as it can and displays a message indicating that some of the characters you entered have not been stored.

DATE Fields

You enter calendar dates in **DATE** fields in the form `[mm]m[d]d[yy]yy`, with any non-numeric characters as optional dividers between the month, day, and year parts of the date. For example, for August 4, 1985, you could enter `Aug 4 1985`, `08/04/85`, `8.4.85`, or `08 04 1985`. (If you enter a two-digit year, **PERFORM** assumes it is a year in the 1900's.) After you press **RETURN**, **PERFORM** displays the date in the format specified in the instructions used to create the form.

MONEY Fields

You enter dollars-and-cents amounts in **MONEY** fields. **PERFORM** displays money amounts with a dollar sign and decimal point. When you enter data in a **MONEY** field, do not enter a dollar sign or comma.

To enter four thousand dollars in a **MONEY** field, type `4000` or `4000.00`; **PERFORM** displays `$4000.00`.

Confirming Your Entry

When you have entered the data you want for a particular field, press **RETURN** or **TAB**. This tells **PERFORM** to accept the data you entered and move the cursor to the next field.

Data Checking in PERFORM

PERFORM does not allow you to enter data of the wrong type into a field. For example, **PERFORM** does not accept alphabetic characters in a numeric field. If you enter invalid data, **PERFORM** displays

Error in field

on the Status Line at the bottom of the screen. You cannot move the cursor out of a field that contains an invalid value. You must enter an acceptable value or press the program interrupt key to cancel the **Add** option.

When you create a form, you can specify acceptable values for a particular field. For example, you can specify that a numeric field only accept numbers from 1 through 50 or that a character field only accept one of five particular words. If you enter data the form cannot accept, **PERFORM** displays

This value is not among the valid possibilities

on the Status Line.

You can also create a special format—called a *picture*—for a field. For example, social security numbers always have the same format, and you might specify that format in a picture for a social security number field. You cannot enter data that does not conform to the picture for a field. (For details, refer to Chapter 3, “The FORMBUILD Screen Form Generator,” in the *INFORMIX-SQL Reference Manual*.)

Special Functions

As you enter data or a query, you can access three special functions by using specific keys.

Function	Key Used
-----------------	-----------------

Help	The CTRL-W key (and F1 on DOS systems) displays a HELP Screen containing a short summary of special keys, control keys, editing keys, and other information about PERFORM .
-------------	---

Execute	The ESC key runs the option you select. To add a new row, type a to select the Add option, enter the information for the row, and press ESC to add the row to the database.
----------------	--

Interrupt	On most UNIX systems, the DEL key interrupts or cancels the option you are using. For example, if you select Add when you really want Query , press DEL , then select the Query option.
------------------	--

On DOS systems (and some UNIX systems), the **CTRL-C** key interrupts or cancels the option you are using.

Positioning the Cursor

The keys used to position the cursor on the screen follow:

Movement	Key Used
-----------------	-----------------

Next Field	The RETURN and DOWN ARROW keys move the cursor to the next field.
-------------------	---

Backspace	The BACKSPACE and LEFT ARROW keys move the cursor backward one character at a time without erasing any text. Pressing either key at the beginning of a field moves the cursor to the previous field.
------------------	--

Forward	The RIGHT ARROW key moves the cursor forward one character at a time without erasing any text. Pressing the RIGHT ARROW key at the end of a field moves the cursor to the next field.
Fast Forward	The CTRL-F key (F6 on DOS systems) moves the cursor down the screen rapidly, stopping in the first field on each line. Use CTRL-F when you want to move quickly to the bottom of a form that contains many fields.
Fast Backspace	The CTRL-B key (F5 on DOS systems) moves the cursor up the screen rapidly, stopping at the last field on each line. Use CTRL-B when you want to move quickly to the top of a long form.

Field Editing

If you make a mistake while entering data in a field, you can correct it by backspacing and retyping. However, it is often faster to use the **PERFORM** field-editing feature. You can use two editing modes to enter data into a field:

- In *typeover mode*, the characters you type replace existing data. You could use typeover mode to change “Sports ’R Us” to “Abe’s Sporting Goods.”
- In *insert mode*, the characters you type push existing data to the right. You could use insert mode to add an *i* to *Rchard*.

When you access **PERFORM**, it assumes you want to be in typeover mode; you must press **CTRL-A** to get into insert mode. Press **CTRL-A** a second time to return to typeover mode.

Edit data that appears in a field by using the following keys:

Function	Key Used
Backspace	The BACKSPACE and LEFT ARROW keys move the cursor back one character at a time without erasing any text. If you press either key at the beginning of a field, the cursor moves back to the previous field.
Forward	The RIGHT ARROW key moves the cursor forward one character at a time without erasing any text. If you press RIGHT ARROW at the end of a field, the cursor moves forward to the next field.
Delete a Character	The CTRL-X key deletes the character beneath the cursor. The cursor remains in place, and text shifts over to fill the space that was occupied by the deleted character.
Change Mode	The CTRL-A key (Ins on DOS systems) shifts between insert and typeover mode. When you access PERFORM , you are in typeover mode.
Delete Forward	The CTRL-D key (F9 on DOS systems) deletes everything from the current cursor position to the end of the field you are editing.
Repeat Data	<p>On UNIX systems, the CTRL-P key enters the most recently displayed value in a field. When you use the Add option to enter several rows in which one or more fields contain the same data, you can avoid retyping the data by pressing CTRL-P. When you use the Update option, CTRL-P restores the value that appeared in a field before you modified it.</p> <p>On DOS systems, the F3 key enters the most recently displayed value in a field.</p>

Clear Screen The CTRL-C key clears any search criteria you have entered with the **Query** option.

Storing the Row

When you have entered information in all the appropriate fields, tell **PERFORM** to store this information in the active table by pressing the **ESC** key. To cancel the **Add** option without storing the new row data, press the program interrupt key.

When you press the **ESC** key, **PERFORM** adds the new row information to the active table and displays

Row added

Changing Existing Data

Use the **Update** option to change the contents of an existing row. For example, you would use **Update** to enter a new phone number for a customer in the **customer** table.

You can **Update** only the current row—the row currently appearing on the screen. You use the **Query**, **Next**, and **Previous** options to display the row you want to update. You will learn how to use these options in the next chapter.

When the row you want to update appears on the screen, you update it in the following way:

1. Type **u** to select the **Update** option. **PERFORM** displays the **UPDATE** Menu, and the cursor moves into the first field on the form.
2. Use the cursor-positioning keys to move the cursor from field to field and the editing keys to change the data in as many fields as you like.
3. Press **ESC** to store the changes in the active table. If you do not want to save the changes, press the program interrupt key to cancel the **Update** option. The row remains unchanged.

When you press the **ESC** key, **PERFORM** modifies the row to reflect the changes you made and displays the following message:

This row has been changed

Removing a Row

Use the **Remove** option to delete a row from the active table. For example, you would use **Remove** to delete information for a customer who quits the sporting goods business.

Like **Update**, the **Remove** option affects only the row that currently appears on the screen.

When the row you want to delete appears on the screen, you can delete it in the following way:

1. Type **r** to select the **Remove** option. **PERFORM** presents the **REMOVE** Menu and, in the upper left corner of the screen, asks

```
REMOVE: ☐ Yes ☐ No
Removes this row from the active table.
```

2. To remove the row, enter **y**. If you do not want to remove the row, enter **n**.
3. Enter **y** and **PERFORM** removes the row, clears the information from the screen, and displays the following message:

```
Row deleted
```

Exiting PERFORM

When you are finished working with a form, use the **Exit** option to leave **PERFORM** and return to the **FORM Menu**. You can then choose to work with another form or return to the **INFORMIX-SQL Main Menu**.

If you accessed **PERFORM** from the operating system command line, the **Exit** option returns you to the operating system.

Chapter Summary

- You can use a **PERFORM** screen form to add, update, and remove database data. A screen form is like a printed form, with blank spaces you fill in. These spaces are called fields.
- A database can have any number of screen forms. Each screen form can include data from one or several tables.
- To work with **PERFORM**, select the **Run** option on the FORM Menu.
- Use the **Add** option to enter data into the database.
- You can change data with the **Update** option and remove rows with the **Remove** option.
- Within the **PERFORM** program, pressing the CTRL-W key (and the F1 key on DOS systems) displays a HELP Screen, the ESC key executes the option you request, and the DEL or CTRL-C key interrupts or cancels an option.
- You exit **PERFORM** with the **Exit** option.

Chapter 4

Querying a Database

Chapter 4 Table of Contents

Chapter Overview	5
What Is a Database Query?	6
Searching for an Exact Match	7
Searching for a Group of Rows	9
The Current List	10
The Next and Previous Options	11
Finding All the Rows in a Table	12
How Database Queries Work	13
Searching with Relational Operators	14
Numeric Fields	15
DATE Fields	15
MONEY Fields	15
CHAR Fields	16
Using Wildcard Characters	16
Searching for a Range of Values	17
Searching for the Highest and Lowest Values	18
Searching for Alternate Values	19
Query Operators and Short Fields	19
Searching in SMALLFLOAT and FLOAT Fields	20
Example Database Queries	21
Moving from Screen to Screen	22
Saving the Results of a Query	23
Chapter Summary	27

Chapter Overview

This chapter explains how to use a screen form to retrieve data stored in an **INFORMIX-SQL** database. The following topics are discussed:

- Conducting database queries
- Using search criteria in a database query
- Searching for a single row
- Searching for a group of rows
- Retrieving all rows in a table
- Scanning through a group of rows
- Moving from page to page on a multiple-page form
- Saving the results of a query in a file

You should read this chapter if you want to learn how to use **PERFORM** to query a database.

For complete information about querying a database with **PERFORM**, refer to Chapter 4, “The **PERFORM** Screen Transaction Processor,” in the *INFORMIX-SQL Reference Manual*.

What Is a Database Query?

Once you store some data in a database, you can use a screen form to inquire about the data. You can find out how many rows of data exist in a table, and you can select and display the particular row or rows you want to review.

This process is called *querying* a database. **PERFORM** can retrieve and organize your data in many different ways, depending on the screen form you use and how you structure your query.

The examples in this section are based on the **customer** form included with the demonstration database. The form, as it appears on the screen, is displayed in Figure 4-1.

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table. ** 1: customer table**

CUSTOMERS

Customer Number: []

Company : []

First Name: [] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

Figure 4-1. The **customer** Form

Searching for an Exact Match

The simplest way to query a database is to ask **PERFORM** to locate a single row. To search for a row, select the **Query** option and then enter data on the screen form the way it appears in the row you want **PERFORM** to find.

The data you enter is called *search criteria* because **PERFORM** uses this information to select rows as it searches the database. Some database management systems refer to this approach as *query by example*.

You can use the **customer** form to search for an individual customer's row. First, type **q** to select the **Query** option, and then type the company's name in the Company field. For example, to search for the row containing information about the "All Sports Supplies" company, you would complete the form like this:

```
QUERY:  ESCAPE queries.  DEL discards query.  ARROW keys move cursor.
Searches the active database table.  ** 1. customer table**
```

CUSTOMERS

Customer Number: []

Company : All Sports Supplies

First Name: [] Last Name: []

Address : []

 []

City : [] State : [] Zip : []

Telephone : []

Figure 4-2. The QUERY Screen in PERFORM (UNIX systems)

The completed QUERY Screen tells **PERFORM** to search for rows in which the company column contains “All Sports Supplies.” Leaving the other fields blank tells **PERFORM** not to consider them as part of the query.

To initiate the search, press the **ESC** key. When **PERFORM** locates the row, it appears on the screen, along with a message on the Status Line indicating the results of the search.

PERFORM: <u>Query</u> Next Previous Add Update Remove Table Screen . . .			
Searches the active database table.		** 1: customer table**	
CUSTOMERS			
Customer Number: [101]			
Company : <u>All Sports Supplies</u>			
First Name: [Ludwig]		Last Name: [Pauli]	
Address : [213 Erstwild Court]			
City : [Sunnyvale]		State : [CA]	Zip : [94086]
Telephone : [408-789-8075]			
1 row(s) found			

Searching for a Group of Rows

PERFORM can also search for a *group* of rows. For example, to find the rows for all customers with stores in Redwood City, complete the **customer** form as follows:

PERFORM: <u>Query</u> Next Previous Add Update Remove Table Screen . . .			
Searches the active database table. ** 1: customer table**			
<hr/>			
CUSTOMERS			
Customer Number: []			
Company : []			
First Name: []		Last Name: []	
Address : []			
City : [Redwood City]		State : []	Zip : []
Telephone : []			
<hr/>			
<hr/>			

The Next and Previous Options

The **Next** and **Previous** options let you scan through the rows in the current list. Type **n** to display the next row in the list, and type **p** to go back to the previous row. When you reach either end of the list, **PERFORM** displays the following message:

There are no more rows in the direction you are going

If you want to skip more than one row at a time, type a number before you enter the option. For example, to skip forward three rows, type **3n**.

Finding All the Rows in a Table

You can locate *all* the rows in the **customer** table (and determine how many there are) by using the **Query** option without filling in any fields on the screen form by typing **q**, and then pressing **ESC**.

The following screen displays the results of the query.

```

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table.                ** 1: customer table**

```

CUSTOMERS

Customer Number: [101]

Company : All Sports Supplies

First Name: [Ludwig] Last Name: [Pauli]

Address : [213 Erstwild Court]

City : [Sunnyvale] State : [CA] Zip : [94086]

Telephone : [408-789-8075]

18 row(s) found

How Database Queries Work

The **Query** option lets you find information in a database. To use it, you specify search criteria by filling in one or more fields on the screen form with the data that appears in the row or rows you want to find.

PERFORM compares each row in the database with the criteria you enter and selects only those rows that satisfy the criteria. **PERFORM** places all the rows it finds that satisfy your search criteria into the current list, displaying the first one on the screen.

If you enter search criteria in more than one field, **PERFORM** only selects rows that satisfy the criteria for all search fields. If you make an error while entering search criteria, you can use **CTRL-C** (UNIX systems only) or **F10** (DOS systems only) to clear the criteria you entered.

When you only want rows in which a field contains a particular value, enter that value in the field. You can also use these other methods:

- The relational and range operators to search for values that lie within a particular range
- *Wildcard* characters to search for a single character or for character strings that match a certain pattern
- Highest-to-lowest value operators to search for the row in which a field has the highest or lowest value

These operators are explained in the next section, “Searching with Relational Operators.” The following table lists the operators, their names, and the data types with which they can be used.

Operator	Name	Data	
		Types	Pattern
=	Equal to	All	= x
>	Greater than	All	> x
<	Less than	All	< x
>=	Greater than or equal to	All	>= x
<=	Less than or equal to	All	<= x
<>	Not equal to	All	<> x
:	Range	all	x:y
	Or	all	x y
*	Wildcard (n≥0)	CHAR	*x, x*, *x*
?	Wildcard (n=1)	CHAR	?x, x?, ?x?
>>	Highest value	all	>>
<<	Lowest value	all	<<

Searching with Relational Operators

The first six operators in the table are *relational operators*; you can use them to describe the acceptable values for a field. They are called relational operators because they describe the relationship between the value you enter and the value or values for which you are looking.

Normally, you do not have to enter the equal to (=) operator; if you do not enter an operator, **PERFORM** assumes you mean “equal to.” For example, when you enter Redwood City in the City field, **PERFORM** assumes you mean =Redwood City.

However, you need to use an equal sign in two situations involving CHAR fields:

- Entering an equal sign by itself in a CHAR field searches for rows in which that field contains NULL values.
- Entering an equal sign followed by an asterisk (*) in a CHAR field searches for rows in which that field contains an asterisk. (You have to use the equal sign with the asterisk because the asterisk is also a query operator.)

Normally, you enter an operator followed by a value, with no spaces between them. The following guidelines explain how to use the relational operators in different types of fields. The examples are based on the **customer** form in the demonstration database.

Numeric Fields

Enter an operator and a number. For example, to search for all rows in which the customer number is greater than 110, enter >110 in the Customer Number field.

DATE Fields

In DATE fields, less than (<) and greater than (>) mean before and after the date you enter, respectively. The **customer** form has no DATE columns; you will have an opportunity to work with DATE columns in the next chapter. For now, recognize that entering <= 10/30/80 in a DATE column locates all records containing October 30, 1980, or earlier in the DATE field.

MONEY Fields

Operators are also useful when querying MONEY fields. To find rows with a value less than \$12,000, enter <12000 in a MONEY field. Do not enter a dollar sign or comma in your query. **PERFORM** adds the dollar sign automatically. A MONEY field is also present in the form that is used in the next chapter.

CHAR Fields

When you use a relational operator in a CHAR field, the following conditions exist:

- Greater than means later in the alphabet, and less than means earlier in the alphabet.
- Lowercase letters are greater than uppercase letters.
- Both uppercase and lowercase letters are greater than numbers.

PERFORM compares characters based on ASCII values. (An ASCII chart appears in Appendix F of the *INFORMIX-SQL Reference Manual*.)

To find rows for customers whose last names are alphabetized after Baxter, enter >Baxter in the Last Name field. Because lowercase letters have a higher ASCII value than uppercase letters, this search criterion also finds *all* names that begin with lowercase letters.

Using Wildcard Characters

The asterisk (*) and the question mark (?) wildcard characters work in queries with CHAR data types. You can use the * to match any group of zero or more characters. For example, to find customers whose last names start with a capital B, enter B* in the Last Name field.

You can use multiple asterisks to search for fields that contain a certain pattern of characters. For example, you might want to search for customers whose retail locations include the words *Avenue* or *Ave*. If you enter *Ave* in the Address field, **PERFORM** finds “41 Jordan Avenue” and “776 Gary Avenue.” The search pattern would also find a truncated address like “41 Jordan Ave.”

You can use the ? to match a single character. The following screen shows a query using the ? wildcard character. **PERFORM** retrieves rows in the **customer** table where the value in the **fname** column contains a four letter “word” beginning with any character and ending with the character string “ick” (“Dick,” “Rick,” “Nick,” and so on).

```

QUERY:  ESCAPE queries.  DEL discards query.  ARROW keys move cursor.
Searches the active database table.           ** 1: customer table**

```

CUSTOMERS

Customer Number: []

Company : []

First Name: [?ick] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

Please enter first name if available

Searching for a Range of Values

The colon (:) is the range operator. It lets you search for all values that lie between one value and another. The range is inclusive.

For example, to search for all zip codes from 94060 through 94080, enter 94060:94080 in the Zip field. When you fill in a field with two values separated by the range operator, **PERFORM** finds rows in which the value of that field lies within the specified range.

For example, to identify stores with customer numbers between 110 and 115, enter 110:115 in the Customer Number field on the **customer** form. Since the range is inclusive, this example finds customers with the numbers 110 and 115, as well as any numbers in between.

You can use the range operator with CHAR fields. Because **PERFORM** compares characters based on ASCII values, lowercase letters are processed after uppercase letters. For example, finding customer names that are alphabetized after (and including) Baxter—but omitting names that begin with a lowercase letter—requires that you enter **Baxter:a** in the Last Name field.

Searching for the Highest and Lowest Values

You can use the **>>** and **<<** symbols in a query to locate the maximum and minimum column values, respectively.

Enter either operator in a field to search the database for the row that contains the lowest or highest value for that field. The **>>** and **<<** symbols work with all data types.

On the **customer** form in the demonstration database, for example, entering **>>** in the Customer Number field searches for the row with the highest customer number.

Searching for Alternate Values

The pipe sign (|) can be used in queries to signify OR. The following screen shows a query using the OR operator.

```
QUERY:  ESCAPE queries.  DEL discards query.  ARROW keys move cursor.
Searches the active database table.          ** 1: customer table**
```

CUSTOMERS

Customer Number: [110|118]

Company :

First Name: [] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

Upon executing this query, **PERFORM** searches for rows where the value in the **customer_num** column equals either 110 or 118. The row(s) returned from this query are placed in the Current List.

Query Operators and Short Fields

Occasionally a display field may be too short to hold the search criterion you enter. When this occurs, **PERFORM** creates a work space at the bottom of the screen; when you press **RETURN**, the work space disappears. The field contains the criterion you entered in the work space, even though you can only see the portion that fits in the field.

For example, when you enter 94060:94080 in the Zip field, you see the following screen:

```

QUERY:      ESCAPE queries.  DEL discards query.  ARROW keys move cursor.
Searches the active database table.              ** 1: customer table**

```

CUSTOMERS

```

Customer Number:  [          ]

    Company : [          ]
    First Name: [          ]    Last Name:  [          ]

    Address  : [          ]
                [          ]

    City    : [          ]    State  :  [    ]    Zip   :  [94060]

    Telephone: [          ]

```

[94060:94080]
]

Searching in SMALLFLOAT and FLOAT Fields

The number **PERFORM** displays in a **SMALLFLOAT** or **FLOAT** field may differ slightly from the number you enter, due to the way in which **INFORMIX-SQL** stores **FLOAT** and **SMALLFLOAT** numbers. This difference can make searching for an exact match in a floating-point field difficult. For example, if you enter 1.1 in a floating-point field, **PERFORM** may actually search for 1.10000001. You can solve this by searching for a range of values, such as 1:1.2.

Small Machine Features: On some small machines, SMALLFLOAT and FLOAT numbers are treated as DECIMAL numbers.

Example Database Queries

In the following queries, you can verify the results displayed on the screen by referring to the **customer** table in Appendix A.

To find rows for customers located in Menlo Park, type **q** to select the **Query** option. Enter **Menlo Park** in the City field, and then press **ESC** to begin the search.

To search for information on customers located in Menlo Park with a customer number greater than 110, enter **Menlo Park** in the City field, **>110** in the Customer Number field, and press **ESC**.

To find customers located in Redwood City with a customer number between 101 to 110 and a zip code of 94062, query the **customer** form as follows:

1. Enter **101:110** in the Customer Number field.
2. Enter **Redwood City** in the City field.
3. Enter **94062** in the Zip field.

Moving from Screen to Screen

Just as printed forms sometimes fill several pages, screen forms may take up more than one screen “page.” You might create a multiple-page form to include a great deal of data or simply because it makes better visual or organizational sense to do so.

When a form contains more than one page, you use the **Screen** option to move back and forth between pages.

The **Screen** option moves you to the next page of a form. When you reach the last page, the **Screen** option takes you back to page 1.

You can type a number before the **Screen** option to select the particular page of the form that you want; **PERFORM** advances directly to that page. For example, to move to the fourth page of a multiple-page form, you type 4s.

The **sample** form included in the demonstration database is a three-page form: you can use the **Screen** option with this form.

Saving the Results of a Query

Even though the **Query** option provides rapid access to the information in a database, you may occasionally want to keep a row of query results permanently. To do this, you use the **Output** option. The **Output** option writes the results of a query to a file. You can create a new file or you can append the data to an existing file.

Although you cannot use this file with other components of **INFORMIX-SQL**, the **Output** option is particularly useful when you want to print data from one or more rows. First use the **Output** option to create a file, and then print it as you would any other text file.

Output produces the current list only. Before you can select the **Output** option, you must first use the **Query**, **Next**, or **Previous** options to locate the row or rows you want to output.

Output only produces a copy of the displayed page of the screen form. When using a single-page form, this presents no problem. If you want to output data from several pages of a multiple-page form, however, you must output each page separately.

For example, you can use the **customer** form to create a list of customers located in Palo Alto. Select the **Query** option; enter Palo Alto in the City field, then press the **ESC** key.

The results of the query look like the next screen.

```
PERFORM: Query  Next  Previous  Add  Update  Remove  Table  Screen . . .
Searches the active database table.                ** 1: customer table**

-----
                                CUSTOMERS
-----
Customer Number:  [ 103          ]

  Company :  Phil's Sports
First Name:  [ Philip          ]  Last Name:  [ Currie          ]

  Address :  [ 654 Poplar      ]
             [ P. O. Box 3498  ]

    City :  [ Palo Alto        ]  State :  [ CA          ]  Zip :  [ 94303        ]

Telephone :  [ 415-328-4543    ]

-----
2 row(s) found
```

Rows for the two Palo Alto businesses are now in the current list. Type o to select the **Output** option. In the upper left corner of the screen, **PERFORM** displays

```
Enter output file (default is perform.out):
Enter an output file name.
```

The default file is the file you most recently used with the **Output** option; if you have not used the **Output** option since accessing **PERFORM**, the default is **perform.out**. You can enter the name of the file you want **PERFORM** to use and then press **RETURN**, or just press **RETURN** to use the default file name. By default, the file appears in your current working directory. You can place the file in another directory by providing a full pathname.

When you press the **RETURN** key, **PERFORM** displays the name of the file you select and asks if you want to create a new file or append the output to an existing file. If you choose the default file name **perform.out**, you will see

```
FORM OUTPUT FILE:  Append  Create
Adds new data to an existing output file.
```

If you select a file that already contains data and you do not want to delete that data, type **a** to add the new data to the end of the existing file. To create a new file, type **c**. If you select an existing file and select the **Create** option, **PERFORM** deletes the old version of the file when you run the **Output** option, and you lose any data previously stored in the file.

When you press the **RETURN** key, **PERFORM** displays this message:

```
FORM OUTPUT FILE LIST:  Current-list  One-page
Writes the current list to the file.
```

You can ask **PERFORM** to output data for all rows in the current list, or you can request a single page at a time. Type **o** to output only the row that currently appears on the screen. Type **c** to output all rows in the current list.

To output all pages of a multiple-page form, you must alternate between the **Output** and **Screen** options. To output some, but not all, of the rows in the current list, you can alternate between the **Output** and **Next/Previous** options.

When you select either the *current-list* or the *one-page* option, **PERFORM** starts writing rows to the file you selected, displaying a counter at the bottom of the screen:

```
Output record number 1
```

If you select the **Current-list** option, **PERFORM** displays each page that it outputs and updates the counter as it writes rows to the file.

This is what the **perform.out** file, which is based on your search for Palo Alto businesses, looks like when printed:

CUSTOMERS

Customer Number: [103]

Company : Phil's Sports
First Name: [Philip] Last Name: [Currie]

Address : [654 Poplar]
[P. O. Box 3498]

City : [Palo Alto] State : [CA] Zip : [94303]

Telephone : [415-328-4543]

CUSTOMERS

Customer Number: [107]

Company : Athletic Supplies
First Name: [Charles] Last Name: [Ream]

Address : [41 Jordan Avenue]
[]

City : [Palo Alto] State : [CA] Zip : [94304]

Telephone : [415-356-9876]

Chapter Summary

- The search criteria you enter on a screen form determine the rows that are returned in a database query.
- You can search for an exact match, or you can use operators to specify the particular row or rows you want.
- **PERFORM** stores the results of each query in a temporary space called the current list.
- You use the **Next** and **Previous** options to scan through the current list.
- You use the **Screen** option to move to the next page of a multiple-page screen form.
- You can save the results of a query in a system file using the **Output** option. This is particularly useful if you want to print data from one or more rows in the current list.

Chapter 5

Using Multiple-Table Forms

Chapter 5 Table of Contents

Chapter Overview	5
What Is a Multiple-Table Form?	6
The Active Table	7
Changing the Active Table	7
Example	8
Join Fields	11
The orderform Form	12
Verify Joins	14
Lookup Joins	15
The Master-Detail Relationship	16
Selecting the Detail Table	16
Returning to the Master Table	19
Multiple Tables Mean Multiple Current Lists	20
The Current Option	20
Chapter Summary	21

Chapter Overview

This chapter describes the special features of **PERFORM** that are available on forms that include data from more than one table. The discussion describes the following topics:

- Multiple-table forms
- The active table
- Join fields
- Verify joins
- Lookup fields
- Master-detail relationships
- Multiple-screen forms

You should read this chapter if you will be using or creating screen forms that include data taken from more than one table.

What Is a Multiple-Table Form?

A form that includes data from more than one table is called a *multiple-table form*. The maximum number of tables that can be included in the form depends upon the open-file limitation on your system. The tables must all be part of the same database.

Two **PERFORM** features are available only with multiple-table forms:

- **Join Fields**, in which you can use a single field to represent data from columns in different tables.
- **Master-Detail Relationships**, in which, once you join fields, you can create master-detail relationships that make it possible to conduct cross-table queries with a single keystroke.

This chapter presents important background information on the **PERFORM** program, and then discusses these techniques in detail.

The Active Table

PERFORM assigns a sequential number to each database table represented on a screen form. The order in which **PERFORM** assigns these numbers is determined when you create the form.

When you access **PERFORM**, it selects the first table as the *active table*. Its name and number appear on the second Information Line, and only its fields are surrounded by brackets (or the alternate field delimiters you have selected).

Changing the Active Table

You can use the **Table** option on the **PERFORM** Menu to select the table with the next sequential number as the new active table. When you do, **PERFORM** updates the name and number on the Information Line, and shifts the brackets on the screen to the fields in the next table. When the last table is the active table, selecting the **Table** option selects the first table again.

The **Table** option automatically selects and displays whichever page of the screen form contains the greatest number of fields from the new active table.

To use the **Table** option, type t. If you know the sequential number that **PERFORM** has assigned to the table you want (the number that appears on the Information Lines), type it before you select the **Table** option. For example, you select the third table by typing 3t.

Example

The demonstration database contains five tables:

- The **customer** table contains one row for each customer.
- The **orders** table contains one row for each order placed by a customer.
- The **items** table contains one row for each item ordered in a customer order.
- The **stock** table contains one row for each item of stock in the store's inventory.
- The **manufact** table contains one row for each manufacturer who does business with the store.

The **orderform** form includes data from four of these tables: **customer**, **orders**, **items**, and **manufact**. When you use the **orderform** form with **PERFORM**, you see the following screen:

```

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches for the active database table.          ** 1: customer table**
-----
CUSTOMER INFORMATION:
Customer Number: [          ] Telephone: [          ]
Company: [          ]
First Name: [          ] Last Name: [          ]
Address: [          ]
City: [          ] State: [          ] Zip: [          ]
-----
ORDER INFORMATION:
Order Number: [          ] Order Date: [          ]
Stock Number: [          ] Manufacturer: [          ]
Quantity: [          ]
Total Price: $ [          ]
SHIPPING INFORMATION:
Customer P.O.: [          ]
Ship Date: [          ] Date Paid: [          ]

```

The second Information Line indicates that **customer** is Table 1 and the active table. Fields from the **customer** table (on the top half of the screen) appear in brackets. If you use the **Table** option to select the next table, this is what you see:

```

PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Selects the current table.          ** 2: orders table**
-----
CUSTOMER INFORMATION:
Customer Number: [          ] Telephone: [          ]
Company: [          ]
First Name: [          ] Last Name: [          ]
Address: [          ]
City: [          ] State: [          ] Zip: [          ]
-----
ORDER INFORMATION:
Order Number: [          ] Order Date: [          ]
Stock Number: [          ] Manufacturer: [          ]
Quantity: [          ]
Total Price: $ [          ]
SHIPPING INFORMATION:
Customer P.O.: [          ]
Ship Date: [          ] Date Paid: [          ]

```

The Information Line shows that **orders** is now the active table, and fields from the **orders** table (on the lower half of the screen) are in the brackets.

Note that the Customer Number field remains in brackets when either the **customer** table or the **orders** table is the active table. The “Join Fields” section that follows discusses the significance of this.

Finally, when you use the **Table** option to select the next table, this is what you see:

```
PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Selects the current table.                ** 3: items table**
-----
CUSTOMER INFORMATION:
Customer Number:                          Telephone:
Company:
First Name:                              Last Name:
Address:

City:                                    State:      Zip:
-----
ORDER INFORMATION:
Order Number: [          ]      Order Date:
Stock Number: [          ]      Manufacturer: [    ]

Quantity: [          ]
Total Price: [$          ]

SHIPPING INFORMATION:
Customer P.O.:

Ship Date:                              Date Paid:
```

The second Information Line shows that **items** is now the active table, and fields from the **items** table (on the lower half of the screen) are in brackets. The Order Number field remains in brackets when either the **orders** table or the **items** table is the active table.

Note: Although the **manufact** table contributes information to the screen form, it is never available as the active table; it is never available for querying, updating, or adding new information.

Join Fields

Use a multiple-table form whenever you want to have access to the data in more than one table at the same time. Normally the tables contain at least one column that contains the same kind of information.

When you create a multiple-table form, you join columns from different tables that contain the same information. In **PERFORM**, *joining* means using a single field to represent data taken from more than one table. The display field is called a *join field*, and the columns involved are called *join columns*.

A join field represents data from more than one table and appears in brackets whenever any of the tables containing the data are active. Think of the join field as a bridge that links the related information in two or more different tables.

For example, the **customer__num** column in the **customer** table and the **customer__num** column in the **orders** table both contain customer numbers. Similarly, the **order__num** column in the **orders** table and the **order__num** column in the **items** table both contain order numbers.

As illustrated in the previous section, the Customer Number field is in brackets when either the **customer** table or the **orders** table is the active table. Similarly, the Order Number field remains in brackets when either the **orders** table or the **items** table is the active table.

To review the next customer row, use the **Next** option to advance to the next row in the current list.

```
PERFORM: Query [Next] Previous Add Update Remove Table Screen . . .
Shows the next row in the Current List.      ** 1: customer table**
-----
CUSTOMER INFORMATION:
Customer Number: [102] Telephone: [415-822-1289]
Company: [Sports Spot]
First Name: [Carole] Last Name: [Sadler]
Address: [785 Geary St]
City: [San Francisco] State: [CA] Zip: [94117]
-----
ORDER INFORMATION:
Order Number: Order Date:
Stock Number: Manufacturer:
Quantity:
Total Price: $
SHIPPING INFORMATION:
Customer P.O.:
Ship Date: Date Paid:
```

The Sports Spot company is customer 102. The join field lets you see information about orders placed by this customer at the bottom of the screen. In this instance, the Sports Spot has yet to place an order, so the bottom half of the screen is blank.

Go ahead and press the n key two more times. Notice how the information about the customer and that customer's order(s) changes each time n is pressed.

When you reach customer number 104, your screen looks like this:

```
PERFORM: Query [Next] Previous Add Update Remove Table Screen . . .
Shows the next row in the Current List.          ** 1: customer table**
-----
CUSTOMER INFORMATION:
Customer Number: [104] Telephone: [415-368-1100]
Company: [Play Ball!]
First Name: [Anthony] Last Name: [Higgins]
Address: [East Shopping Cntr.]
          [422 Bay Road]
City: [Redwood City] State: [CA] Zip: [94026]
-----
ORDER INFORMATION:
Order Number: 1001 Order Date: 01/20/1986
Stock Number: 1 Manufacturer: HRO
                      Hero
                      Quantity: 1
                      Total Price: $250.00
SHIPPING INFORMATION:
Customer P.O.: B77836
Ship Date: 02/01/1986 Date Paid: 03/22/1986
```

Verify Joins

It is often useful to verify that data exists in one table before **PERFORM** accepts it as input to another. For example, while entering an order on the **orderform** form, you do not want to enter a nonexistent customer number.

A special type of join, called a *verify join*, ensures that you can only enter data that already exists in a database table. You set up verify joins when you create a screen form.

If you enter invalid data in a verify join field, **PERFORM** displays an error message. For example, if you enter an invalid customer number while **orders** is the active table, **PERFORM** displays

This is an invalid value -- it does not exist in "customer" table

You can read about how to create verify joins in Chapter 6, "Creating Your Own Forms."

Lookup Joins

Occasionally you may want **PERFORM** to display data that is somehow related to the data you enter on the form. For example, for reference purposes it might be useful for **PERFORM** to display a manufacturer's full name when you enter the manufacturer code.

You can use the lookup feature to find and display data. The fields in which this data appears are called *lookup fields*, and you can never enter data (or search criteria) into them.

For example, the second line of the ORDER INFORMATION section of the **orderform** form shows the stock number and manufacturer code for each item ordered. The manufacturer code is part of the **items** table, but the manufacturer name is stored only in the **manufact** table; **PERFORM** looks it up when you enter the manufacturer code in the field labeled "Manufacturer." You can read about how to create lookup joins in Chapter 6.

Using the lookup feature to display information from a table does not make that table available for entering data or conducting queries. The **orderform** form shows information from the **manufact** table, but the information appears in a lookup field. You cannot make **manufact** the active table; you cannot query it, and you cannot enter, delete, or update the data stored in it.

The Master-Detail Relationship

When a form includes at least one join field, you can also define a master-detail relationship. *Master-detail* refers to the relationship between two tables represented on a screen form. The master-detail relationship simplifies queries that involve data from several tables.

In a master-detail relationship, one table is the master table, and the other is the detail table. You use the **Master** and **Detail** options to switch between tables and to conduct automatic database queries.

There can be several master-detail relationships defined for the same form. A table may have several detail tables but only one master table.

You create a master-detail relationship by including specific instructions in the form specification. This process is described in Chapter 6, “Creating Your Own Forms.”

Selecting the Detail Table

When you use the **Detail** option, **PERFORM** selects the detail table of the currently active table as the new active table and automatically queries the detail table, using the value displayed in the join field as the search criterion.

Type **d** to select the detail table and initiate the search. **PERFORM** displays

Searching ...

while conducting the search and then indicates the results of the query:

4 row(s) found

Just as it does with the **Query** option, **PERFORM** places the rows it finds into the current list, which you can scan by using the **Next** and **Previous** options. (If there is already a current list, the **Detail** query replaces it with the new results.)

For example, the **orderform** screen form, with the **customer** table as the active table and customer number 104 as the current row, appears below.

PERFORM: <u>Query</u> Next Previous Add Update Remove Table Screen . . .	
Searches the active database table. ** 1: customer table**	

CUSTOMER INFORMATION:	
Customer Number: [104]	Telephone: [415-368-1100]
Company: [Play Ball!]	
First Name: [Anthony]	Last Name: [Higgins]
Address: [East Shopping Cntr.]	
[422 Bay Road]	
City: [Redwood City] State: [CA]	Zip: [94026]

ORDER INFORMATION:	
Order Number: 1001	Order Date: 01/20/1986
Stock Number: 1	Manufacturer: HRO
	Hero
	Quantity: 1
	Total Price: \$250.00
SHIPPING INFORMATION:	
Customer P.O.: B77836	
Ship Date: 02/01/1986	Date Paid: 03/22/1986

The second Information Line indicates that **customer** is the active table, and the screen shows data about customer number 104.

The **customer** table is the master table for the **orders** table. You can obtain information about the orders placed by customer 104 by using the **Detail** option. When you type d, **PERFORM** switches tables and queries the database, using the value in the Customer Number field as its search criterion.

The following screen is what you see:

```
PERFORM: . . . Current Master Detail Output Exit
Selects a detail table of the current table.      ** 2: orders table**
-----
CUSTOMER INFORMATION:
Customer Number: [ 104      ]      Telephone: 415-368-1100
      Company: Play Ball!
      First Name: Anthony      Last Name: Higgins
      Address: East Shopping Cntr.
                422 Bay Road
      City: Redwood City      State: CA      Zip: 94026
-----
ORDER INFORMATION:
      Order Number: [ 1001      ]      Order Date: [ 01/20/1986 ]
      Stock Number: 1      Manufacturer: HRO
                                Hero
                                Quantity: 1
                                Total Price: $250.00
SHIPPING INFORMATION:
      Customer P.O.: [ B77836      ]
      Ship Date: [ 02/01/1986 ]      Date Paid: [ 03/22/1986 ]

4 row(s) found
```

PERFORM selects the detail table—**orders**—and automatically initiates a query based on the join field—the one labeled “Customer Number.” **PERFORM** found rows for four orders placed by customer 104. You can now use the **Next** and **Previous** options to scan these rows.

If you use the **Detail** option when no detail table exists for the active table, **PERFORM** displays

No detail table has been specified for this table

If no explicit master/detail relationship exists, **PERFORM** displays an error message when you use the **Master** option or the **Detail** option without a table number.

Returning to the Master Table

The **Master** option returns you to the master table after you use the **Detail** option to conduct a query. The **Master** option does not initiate a database query; it simply returns you to the table that was active before you entered the **Detail** option.

Type m to return to the master table. **PERFORM** selects the master table. If you use the **Master** option when there is no master table for the active table, **PERFORM** displays

No master table has been specified for this table

Multiple Tables Mean Multiple Current Lists

PERFORM maintains a separate current list for each table included on a screen form. This means you can query one table, switch tables, query the second table, then return to the first table and still use the **Next** and **Previous** options to review the results of the original query.

The Current Option

If your form includes a join field, you may occasionally lose your place in one of the current lists. Should this occur, you can use the **Current** option to return to the current position in the current list.

The **Current** option also serves a second purpose. On multiuser computer systems, more than one person can use a database at the same time. This means that another user could modify a row while it appears on your screen.

When you run the **Current** option, **PERFORM** rereads the row from the database and displays the most up-to-date information. If you suspect someone may have changed the currently displayed row, just enter the **Current** option.

Chapter Summary

- A multiple-table screen form contains data from more than one table in a database.
- One table is always the active table. The Information Lines always show the name of the active table and the sequential number assigned to it by **PERFORM**.
- The **Table** option selects the next table as the active table.
- A join field represents data taken from more than one table and appears surrounded by brackets (or alternate delimiters) whenever any of them is the active table.
- You can specify verify joins when you create a screen form. **PERFORM** will ensure that certain data exists in one table before accepting it as input to another table.
- You can specify lookup joins when you create a form. **PERFORM** will display data from a table based on the data you enter on the form.
- Any form that includes a join field may also include one or more master-detail relationships.
- You can use the **Detail** option on the **PERFORM** Menu whenever join fields exist on the form. Use this option to automatically query the detail table using the value displayed in the join field as the search criterion.
- **PERFORM** maintains a separate current list for each table included on a form.

Chapter 6

Creating Your Own Forms

Chapter 6 Table of Contents

Chapter Overview	5
What Is a Form Specification?.....	6
Default Form Specifications	6
How to Create a Default Form.....	7
How to Modify an Existing Form.....	7
How to Create a Form	8
Naming the Form.....	9
Choosing the Tables to Include in the Form.....	10
Compiling the Default Form Specification	11
If There Are Errors.....	12
PERFORM Uses Two Files	13
What a Default Form Looks Like.....	13
An Example Default Form.....	14
How to Modify an Existing Form.....	15
What the Form Specification Contains.....	16
The Example Form Specification	17
Database Section.....	18
Screen Section	18
Tables Section	19
Attributes Section	19
The customer Form Specification	20
Comparing the custom and customer Forms.....	21
Database and Tables Sections	22
Screen Section	22
Attributes Section	23
Using FORMBUILD Attributes.....	24
How to Enter Attributes	25
The Field Tag	25
The Equal Sign.....	25
The Column Name.....	25
The Attribute List.....	26
Using Blank Space	26
Displaying Comments on the Screen.....	26
Changing Data from Lowercase to Uppercase	27
Changing Default Field Values.....	28

Using Today’s Date as the Default 29

Requiring Input..... 30

Specifying Acceptable Values 31

 The Lowest Value Must Appear First 31

 INCLUDE with CHAR Fields 32

Verifying Input 33

Joining Fields 33

 Verify Joins 37

 Lookup Joins 38

Instructions Section 40

Chapter Summary..... 41

Chapter Overview

Earlier chapters described how to enter data and query a database using existing screen forms. This chapter explains how to create your own forms. The following topics are considered:

- Creating a default form specification
- Using a default screen form
- Modifying a default form specification

You should read this chapter if you want to learn how to create and customize your own screen forms.

For complete information about the features introduced in this chapter, refer to Chapter 3, “The FORMBUILD Transaction Form Generator,” in the *INFORMIX-SQL Reference Manual*.

What Is a Form Specification?

Creating a screen form is a two-step process. First, you create a *form specification*, and then you compile it. A form specification is a system file containing instructions that describe how you want the form to look and what database information you want it to access. Compiling converts the instructions in a form specification file into a format that **INFORMIX-SQL** can use.

You can create a form specification from scratch using a system editor. One way to do this is to use the **New** option on the FORM Menu. Selecting the **New** option calls the system editor and places you in a new editing buffer. But it is often easier to first create a *default form specification* using the **Generate** option on the FORM Menu. You can then modify the default form specification to meet your specific requirements.

Default Form Specifications

When you create a default form specification, **INFORMIX-SQL** asks you which tables contain the data you want to use with the form. **INFORMIX-SQL** then creates a default form specification.

The default specification contains the basic instructions you will need to create the form. You can use the default specification as it is, but you need to modify it if you want to take advantage of the more powerful features of **PERFORM**.

How to Create a Default Form

The **Generate** Option on the FORM Menu is used to create a default form specification. **INFORMIX-SQL** asks for the name of the table or tables you want to use with the form, creates a default form specification file, and then compiles the specification.

Once the specification file is compiled, you can use the form in **PERFORM**. You repeat these steps for each form you create.

How to Modify an Existing Form

If you want to change the form, you will need to modify the specification file and recompile it. The FORM Menu includes an option that allows you to do this.

Until you recompile the specification file, any changes made to it will not show up on the screen form itself. This chapter describes in greater detail the process of modifying an existing form.

How to Create a Form

Before creating a screen form, use the **Database** option on the Main Menu to select the database that contains the data you want to use with the form.

After you have selected a database, choose the **Form** option on the Main Menu:

```
INFORMIX-SQL: ☐ Form Report Query-Language User-menu Database Table Exit
Run, Modify, Create, or Drop a form.
..... stores ..... Press CTRL-W for Help .....
```

INFORMIX-SQL displays the FORM Menu. Select the **Generate** option:

```
FORM: ☐ Run Modify ☒ Generate New Compile Drop Exit
Generate a default form.
..... stores ..... Press CTRL-W for Help .....
```

INFORMIX-SQL displays the **GENERATE FORM** Screen:

```
GENERATE FORM >>  
Enter the name you want to assign to the form, then press Return.  
----- stores ----- Press CTRL-W for Help -----
```

Naming the Form

You assign a name to each form at the time you create it. You can use a name up to 8 (in DOS) or 10 (in UNIX) characters long. If you specify the name of a form that already exists, **INFORMIX-SQL** displays

Form with the same name already exists.

and you to must enter a different name.

Enter the name you want to assign to the form, and then press **RETURN**.

Choosing the Tables to Include in the Form

INFORMIX-SQL then asks for the name of the table in the current database containing the information you want to use in the form:

```
CHOOSE TABLE >>☐
Choose the table to be used in the default form.
----- stores ----- Press CTRL-W for Help -----
customer
items
manufact
orders
stock
sysmenuitems
sysmenus
```

Choose a table by typing or highlighting its name and pressing **RETURN**. **INFORMIX-SQL** displays the **GENERATE FORM** Menu:

```
GENERATE FORM: ☐ Table-selection-complete Select-more-tables Exit
Continue creating a default form with the selected tables.
----- stores ----- Press CTRL-W for Help -----
```

The **Generate** option allows you to include data from up to eight different tables on a single form. To use more than one table in a screen form, choose the **Select-more-tables** option. **INFORMIX-SQL** displays the list of available tables and lets you select another table.

When you have chosen all the tables you want to include in the form, select the **Table-selection-complete** option. **INFORMIX-SQL** then creates the default form specification.

Note: If you want to include more than eight tables in a form, create the form from scratch using the **New** option on the Form Menu. See Chapter 3, “The FORMBUILD Transaction Form Generator,” in the *INFORMIX-SQL Reference Manual* for more information.

Use the **Exit** option to return to the FORM Menu if you decide you do not want to generate a default form.

Compiling the Default Form Specification

When you select the **Table-selection-complete** option, **INFORMIX-SQL** creates a default form specification file and compiles the specifications.

You must compile the form specification before you can use the form. Compiling allows **INFORMIX-SQL** to convert the specifications into a format that **PERFORM** can understand. **INFORMIX-SQL** automatically compiles the specification file when you select the **Table-selection-complete** option.

During compilation, messages appear on the screen. When the compilation is successfully completed, the message

Form was successfully compiled

flashes across the bottom of the screen, and you are returned to the FORM Menu.

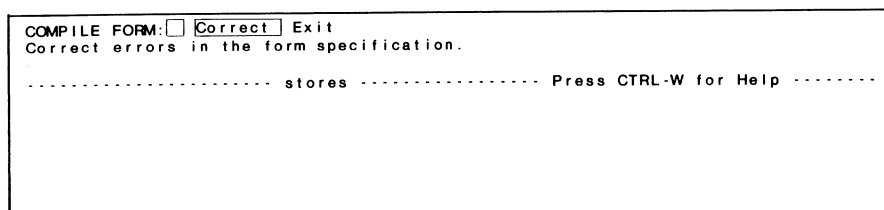
You can now use the form. Later in this chapter, you will learn about the modifications you might want to make to the default form specification file.

If There Are Errors

If you use the **Modify** option to modify a default specification, or the **New** option to create a form specification from scratch, you may accidentally introduce errors. If **FORMBUILD**, the **INFORMIX-SQL** program that compiles form specifications, cannot compile the specification, the message

832: Error(s) found in Form specifications.

appears at the bottom of the screen. **INFORMIX-SQL** displays the **COMPILE FORM** Menu:



The screenshot shows a terminal window with the following text:

```
COMPILE FORM: ☐ Correct ☒ Exit
Correct errors in the form specification.
..... stores ..... Press CTRL-W for Help .....
```

Choose the **Correct** option to correct the errors. When you do, **INFORMIX-SQL** prompts for the name of the system editor you want to use and displays the form specification file within the editor. See Chapter 7 for a discussion on the use of the system editor. The file now contains error messages describing any errors you made.

Note: The text of all error messages, along with suggestions for corrections, is included in a special section at the back of the *INFORMIX-SQL Reference Manual*.

Correct any errors, and then save the specification file. You do not need to delete the error messages themselves; **FORMBUILD** automatically removes them from the file.

When you save the corrected form specification file, **INFORMIX-SQL** redispays the **MODIFY FORM** Menu if you are modifying a form, or the **NEW FORM** Menu if you are creating a form from scratch. You can then recompile the file and generate a usable screen form.

PERFORM Uses Two Files

Once you compile and save a form, two system files are available to **PERFORM**. These files are named with the name you assigned to the form, along with a three-letter extension.

The first file contains the form specifications and is given the extension **.PER** (in DOS) or **.per** (in UNIX). This file contains the instructions that control the appearance and behavior of the form. Use a system editor to modify this specification file if you elect to customize the form for your particular requirements.

The second file contains a compiled version of the form specifications and is given the extension **.FRM** (in DOS) or **.frm** (in UNIX). This file is used by **PERFORM** when you work with a screen form.

INFORMIX-SQL stores these two files in the current directory.

Note: If you create a default form specification from the operating system command line, you can specify a full path-name and have **INFORMIX-SQL** store the two files in that directory. See Appendix M of the *INFORMIX-SQL Reference Manual* for details.

What a Default Form Looks Like

When you create a default form, **INFORMIX-SQL** includes one field for each column in the table or tables you select. The fields are labeled with the column names.

If you select more than one table, each table receives a separate page on the screen form. You will need to use the **Screen** option in **PERFORM** to move from page to page.

An Example Default Form

Follow these steps to create a default form named **custom** that works with the **customer** table in the **stores** database:

1. Use the **Database** option on the Main Menu to select the **stores** database.
2. Select the **Form** option on the Main Menu, followed by the **Generate** option on the FORM Menu.
3. Enter **custom** as the name of the form.
4. Select the **customer** table from the list of tables that INFORMIX-SQL displays.
5. Select the **Table-selection-complete** option.
INFORMIX-SQL compiles the form specification and displays the FORM Menu.

Now you can use the **custom** form with **PERFORM**. Select **Run** on the FORM Menu, and then choose the **custom** form from the list INFORMIX-SQL displays.

Here is what you will see:

The screenshot shows the FORMBUILD utility interface. At the top, there is a menu bar with options: **PERFORM:** Query, Next, Previous, Add, Update, Remove, Table, Screen, and a series of dots. Below the menu bar, the text reads: "Searches the active database table." followed by "** 1: customer table**". The main area displays a list of fields from the customer table, each followed by a set of brackets indicating its structure:

Field Name	Structure
customer_num	[]
fname	[]
lname	[]
company	[]
address1	[]
address2	[]
city	[]
state	[]
zipcode	[]
phone	[]

Figure 6-1. The Default Form Based on the **customer** Table

How to Modify an Existing Form

To change a form that already exists, you need to modify the form specification and then recompile the form. The **Modify** option on the FORM Menu can be used for this purpose.

The next part of this chapter describes the sections that make up a form specification. As you read this material, you can modify the **custom** form to include desired changes. See Chapter 3, "The FORMBUILD Transaction Form Generator," in the *INFORMIX-SQL Reference Manual* for more information on the **Modify** option and the structure of a form specification.

What the Form Specification Contains

Each form specification must include at least four sections:

- Database section
- Screen section
- Tables section
- Attributes section

These four sections appear in every default form specification. A form specification file can also contain an optional fifth section called the Instructions section, but you will not find the Instructions section in a default specification.

The Example Form Specification

The default form specification for the **custom** form appears in Figure 6-2.

```
database stores
screen
{
customer_num      [ f000          ]
fname             [ f001          ]
lname            [ f002          ]
company           [ f003          ]
address1          [ f004          ]
address2          [ f005          ]
city              [ f006          ]
state             [ a0           ]
zipcode           [ f007          ]
phone             [ f008          ]
}
end
tables
customer
attributes
f000 = customer.customer_num;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state;
f007 = customer.zipcode;
f008 = customer.phone;
end
```

Figure 6-2. The **custom** Form Specification

The form specification contains all the information **PERFORM** needs for the **custom** form. It contains Database, Screen, Tables, and Attributes sections; it does not include an Instructions section.

Using the **END** keyword to mark the end of sections in the form specification file is optional. Some users find it helpful to indicate the close of a section with **END**. The forms included with the demonstration database use the **END** keyword.

Database Section

Each form specification begins with a Database section that specifies the database the form is designed to work with. The Database section consists of the **DATABASE** keyword followed by the name of the database.

In a default form specification, the Database section specifies the database that was the current database when you created the specification. In the **custom** specification,

```
database stores
```

indicates that the **custom** form works with the **stores** database.

Screen Section

The Screen section shows how the form appears when you use it with **PERFORM**. However, there are a few differences between the appearance of the form on the screen and in the specification file.

First, the Screen section of the form specification is enclosed within a set of { } braces, and begins with the **SCREEN** keyword. The optional **END** keyword can be used to indicate the conclusion to the section.

Second, the Screen section of the form specification file includes a *field tag* for each field on the form. These tags appear between the brackets that delimit the fields. Each field must have a field tag. **FORMBUILD** assigns field tags automatically when you create a default form specification. In the **custom** specification file, the field tags are **f000**, **f001**, **f002**, and so on.

Except for the field tags, **PERFORM** displays everything between the braces exactly as it appears in the specification.

Tables Section

The Tables section lists the name of each table that contains data used by the form. It appears immediately after the Screen section.

The Tables section in a default form specification lists the table or tables that you chose when you created the specification. In the **custom** specification file,

```
tables
customer
```

indicates that the **custom** form uses data from only one table, the **customer** table.

Attributes Section

The Attributes section explicitly associates each field on the form with a column in one of the tables listed in the Tables section. It begins with the ATTRIBUTES keyword, and lists each field tag, along with the name of the column to which the tag corresponds.

For example, the second line of the Attributes section in the **custom** specification assigns the **f001** field to the **fname** column in the **customer** table:

```
f001 = customer.fname;
```

This statement means that **PERFORM** will display **fname** values in the **f001** field when you use the form to query the database, and store any values you enter in the **f001** field in the **fname** column.

Suggestion: When you create a default form specification, **FORM-BUILD** automatically assigns both table and column names to each field.

In most cases the table name is optional, except in the following instance: you must include the table name when you refer to a column name that appears in more than one of the tables used in the form.

You may want to adopt the `table.column` convention when creating all of your screen forms so that you can be sure that all column references are unambiguous.

The customer Form Specification

You can use the **custom** form to enter and retrieve customer information. Because it is a default form, however, it does not take advantage of the more powerful features of **PERFORM**.

The **customer** form in the **stores** database contains precisely the same information as the default **custom** form, but the two forms look and act differently. Here is the **customer** screen form when used by **PERFORM**:

The screenshot shows a terminal window with the following content:

```
PERFORM: Query Next Previous Add Update Remove Table Screen . . .
Searches the active database table.          ** 1: customer table**
```

CUSTOMERS

Customer Number: []

Company : []

First Name: [] Last Name: []

Address : []

City : [] State : [] Zip : []

Telephone : []

The **customer** form was created by modifying and recompiling the **custom** specification file.

Comparing the custom and customer Forms

The form specification for the **customer** form appears in Figure 6-3. For comparison, Figure 6-4 repeats the form specification for the **custom** form.

```
database
  stores

screen
{
  CUSTOMERS

  Customer Number: [ f000      ]

      Company: [ f001      ]
  First Name: [ f002      ]      Last Name: [ f003      ]

      Address: [ f004      ]
              [ f005      ]

      City: [ f006      ]      State: [ a0      ]      Zip: [ f007      ]

  Telephone: [ f008      ]

}
end

tables
  customer

attributes

f000 = customer_num;
f001 = company, reverse;
f002 = fname, comments = "Please enter first name if available";
f003 = lname;
f004 = address1;
f005 = address2;
f006 = city;
a0   = state, upshift, autonext, include = ("CA", "OR", "NV", "WA"),
      comments = "legal states are CA, OR, NV, or WA";
f007 = zipcode, autonext;
f008 = phone, picture = "###-###-####XXXXXX";

end
```

Figure 6-3. The **customer** Form Specification

```

database stores
screen
{
customer_num      [ f000          ]
fname              [ f001          ]
lname             [ f002          ]
company            [ f003          ]
address1           [ f004          ]
address2           [ f005          ]
city              [ f006          ]
state              [ a0           ]
zipcode            [ f007          ]
phone             [ f008          ]
}
end
tables
customer
attributes
f000 = customer.customer_num;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state;
f007 = customer.zipcode;
f008 = customer.phone;
end

```

Figure 6-4. The **custom** Form Specification

The similarities and differences between the two specifications are discussed in the following sections.

Database and Tables Sections

Both forms are designed to work with the **customer** table in the **stores** database so the Database and Tables sections are the same in both specification files.

Screen Section

The two forms look quite different on the screen:

- Fields on the **custom** form appear in a single column; the **customer** form places them across the full width of the screen.
- The **custom** form labels fields with the names of the columns to which they correspond (**customer_num**,

fname, and so on); the **customer** form uses labels like Name, Address, and Telephone.

- The **customer** form also includes horizontal lines to make the screen more pleasing to the eye.

These differences exist because of differences in the Screen sections of the two form specification files. Using your system editor, you can move fields, change the labels that appear alongside the fields, add horizontal and vertical lines, and make any other changes you want to the appearance of the form.

Be careful not to decrease the widths of the fields when you modify the Screen section, or they may no longer be wide enough to display the data in your database. The width of a display field is the distance between the brackets that surround it.

Attributes Section

In the **custom** specification, the Attributes section simply establishes relationships between field tags and database columns. The **customer** specification assigns different database columns to the field tags and includes attributes that specify how the fields should behave.

Field Tag Assignments. The order in which field tags appear in the Attributes section determines the sequence of fields to which the cursor moves during a **Query**, **Add**, or **Update** action. You can change the order of field tags—or the assignment of database columns to field tags—in the Attributes section to make the form more effective.

For example, the assignment of database columns to field tags in the **custom** and **customer** specifications is different because the layout of fields on the screen is different in each form. In the **customer** form the Company field (corresponding to the **company** column in the **customer** table) appears as the second field in the form, and is assigned field tag **f001**. In the **custom** form the Company field appears as the fourth field in the form, and is assigned field tag **f003**.

Attribute Keywords. The Attributes section of the form specification file assigns each field on the form to a column in one of the tables listed in the Tables section. In the **custom** and **customer** forms, the fields are linked to columns in the **customer** table. For example, in the **custom** form, the line

```
f003 = customer.company;
```

assigns the field **f003** to the column **company** in the **customer** table. The **customer** screen form, however, includes specific *attributes* that define the behavior of individual fields. For example, the **customer** specification assigns the REVERSE attribute to the **f001** field:

```
f001 = customer.company, reverse;
```

The REVERSE attribute causes **PERFORM** to display the field in reverse video (dark letters on a light background).

The next section explains how to use attributes like REVERSE to customize the way your forms look and behave.

Using FORMBUILD Attributes

You can customize each field on a form by adding one or more keywords—called attributes—in the Attributes section. Some attributes, like REVERSE, only affect the way a field appears. Others specify actions that you want **PERFORM** to take when you enter data in the field. For example, by using the UPSHIFT attribute in a CHAR field, **PERFORM** automatically converts any data you enter in the field to uppercase. The UPSHIFT attribute is used in the **customer** form in the State field.

This section explains how to use several more frequently used attributes. Once you understand how to use attributes, you can refer to the *INFORMIX-SQL Reference Manual* for complete information.

How to Enter Attributes

The format you use for entries in the Attributes section is always the same

field-tag = *column-name* [*attribute-list*];

Only the field tag, equal sign, column name, and semicolon are required; an attribute list is optional.

The Field Tag

field-tag is the tag that appears between brackets in the Screen section of the specification. In the **customer** form, these tags are **f000**, **f001**, **f002**, and so on.

The Equal Sign

The equal sign associates the field indicated by the field tag with the column you specify (*column-name*).

The Column Name

column-name is the name of the column whose data you want to appear in the field indicated by the field tag.

If the same column name appears in more than one of the tables in the Tables section, you must enter both the table name and the column name. For example, to refer to the **customer__num** column in the **customer** table, enter

`customer.customer_num`

To refer to the **customer__num** column in the **orders** table, enter

`orders.customer_num`

Default form specifications include the table name for every field in the Attributes section.

The Attribute List

attribute-list may be one attribute or a series of attributes. Separate each attribute from other attributes—and from the column name—by commas. *attribute-list* ends with a semicolon.

The attribute list for a field can occupy more than one line in the form specification. You can break a line at any blank space, except within the text of a comment. (Comments are described in the next section.)

Using Blank Space

You can use blank lines, spaces, and tabs freely in the form specification file, although the text of a comment attribute can not be broken across lines. You do not have to use blank lines in your specification file, but using them makes the instruction easier to read.

Displaying Comments on the Screen

Use the COMMENTS attribute if you want **PERFORM** to display a comment on the Comment Line whenever the cursor moves into a field. For example, you might use a comment to indicate what type of data to enter in a field.

Enter comments, followed by an equal sign and the text of the comment enclosed in quotation marks. The entire text must fit on one line. In the **customer** specification file

```
f002 = fname, comments = "Please enter first name if available
```

causes **PERFORM** to display the comment appearing in quotation marks whenever the cursor moves into the Name field, as shown in the following screen:

QUERY: ESCAPE queries. DEL discards query. ARROW keys move cursor. Searches the active database table. ** 1: customer table**			
CUSTOMERS			
Customer Number: []			
Company : []			
First Name:	[]	Last Name:	[]
Address : []			
City : [] State : [] Zip : []			
Telephone : []			
Please enter first name if available.			

ging Data from Lowercase to Uppercase

The **DOWNSHIFT** and **UPSHIFT** attributes change the case of the data you enter in a **CHAR** field. **UPSHIFT** converts anything you enter to uppercase letters; **DOWNSHIFT** converts entries to lowercase letters. For example,

```
a0 = customer.state,  
upshift;
```

causes **PERFORM** to convert two-letter state abbreviations to uppercase when you enter them into the State field.

Uppercase and lowercase letters have different ASCII values; storing some kinds of CHAR data in one or the other format can simplify sorting data and querying a database. For example, without the UPSHIFT attribute, you might have to search for **CA**, **ca**, **Ca**, and **cA** to find records for customers with locations in California.

The UPSHIFT and DOWNSHIFT attributes are only used in CHAR fields. If you attempt to use them elsewhere, FORMBUILD ignores them.

Changing Default Field Values

PERFORM automatically establishes a default value of NULL for every field on a screen form. When you add data in **PERFORM**, the value you enter into a field replaces its default NULL value. If you do not enter data into a field, **PERFORM** normally stores the NULL default value in the table into which you are adding data.

When you use the **Add** option, **PERFORM** displays the screen form with the following characters in the indicated fields:

Type of Field	Default Display
Character	Blanks
Numeric	Blanks
Date	Blanks
Money	\$

You can avoid NULL values by including the WITHOUT NULL INPUT keywords in the database section of the form specification. When you add a new row to the database, **PERFORM** inserts blanks in CHAR and DATE fields without data and zeros in numeric and MONEY fields without data.

You can use the DEFAULT attribute to replace a field's NULL default value. For example, you might change the default for a phone number field to show your local area code.

Enter **default**, followed by an equal sign and the value you want to assign, as the default value for the field. When you specify a **DATE** value or a **CHAR** value that contains punctuation, symbols, or blank spaces, you must enclose the value in quotation marks.

For example, if most customers are located in California, you might want to set a default for the **State** field:

```
a0 = customer.state,  
    upshift,  
    default="CA";
```

Using Today's Date as the Default

If you enter **default=today** for a **DATE** field, **PERFORM** uses the current date as the default value. In the **orderform** form specification file, the following lines appear in the **Attributes** section:

```
o12 = order_date,  
      default = today;
```

The **Order Date** field on the **orderform** form defaults to the current date.

The **TODAY** attribute is only used in **DATE** fields. If you attempt to use it elsewhere, **FORMBUILD** generates an error message.

Requiring Input

When a column contains essential data, you may want to require a user to enter data for that column. For example, you might want to require a phone number to be entered for each customer. The **customer** screen form specification file would be altered to appear as

```
f008 = phone, required;
```

This would ensure that a telephone number appears in each row of the **customer** table.

When you assign the **REQUIRED** attribute to a field, **PERFORM** requires that you enter data in that field before it accepts a new row for a table. If you try to save a new row without entering data in a required field, **PERFORM** displays

This field requires an entered value

Caution! The **REQUIRED** attribute only works with the **Add** option. You can remove data from required fields with the **Update** option.

Specifying Acceptable Values

Use the **INCLUDE** attribute if you want to specify values that are acceptable as input to a field. **PERFORM** accepts only those values listed with the **INCLUDE** attribute.

Enter **include**, followed by an equal sign and the acceptable values enclosed in parentheses. You can enter one value or a list of values separated by commas, and you can use the **TO** keyword to specify ranges of acceptable values. For example, in the **customer** specification file,

```
a0 = state, upshift, autonext, include = ("CA", "OR", "NV", "WA");  
comments = "legal states are CA, OR, NV, or WA";
```

specifies the values CA, OR, NV, and WA as acceptable values for the **a0** (State) field. The comment line indicates that **PERFORM** accepts no other values.

The Lowest Value Must Appear First

When you specify a range of acceptable values, the lowest value must appear first. You cannot include the range from 50 to 1; you must specify 1 to 50. For example, in the **sample** form included with the demonstration database,

```
i18 = items.quantity, include = (1 to 50),  
comments = "Acceptable values are 1 through 50" ;
```

specifies the values 1 to 50.

You can also specify multiple ranges. For example, the attribute

```
include = (1 to 50, 60 to 100);
```

specifies the values 1 to 50 and 60 to 100.

INCLUDE with CHAR Fields

You can also assign the **INCLUDE** attribute to CHAR fields. For example, you could specify a yes/no field as follows:

```
field-tag = column,  
           upshift,  
           include = (Y, N);
```

You can specify a CHAR range, too. **PERFORM** evaluates ranges based on the ASCII values or the low and high limits of the range. For example,

```
field-tag = column,  
           include (A1 to A5);
```

prevents **PERFORM** from accepting values other than A1, A2, A3, A4, and A5.

When you include a CHAR value that contains punctuation marks, symbols, or blank spaces, you must enclose the entire value in quotation marks:

```
field-tag = column,  
           include = ("sporting goods", "goods, sporting")
```

Suggestion: **PERFORM** displays an error message if you enter a value not specified with the **INCLUDE** attribute:

This value is not among the valid possibilities.

The message does not specify the acceptable values. You may want to use the **COMMENTS** attribute to display a message reminding users of the acceptable values.

Verifying Input

If a field contains particularly critical data, you can use the **VERIFY** attribute to require users to enter the same value twice before **PERFORM** accepts it. That way, your database is less likely to contain incorrect data because of typographical errors.

For example, if you were entering personnel information into a form, you might want to force the user to enter the employee's salary twice before **PERFORM** accepts it. The specification file would include the following lines:

```
f006 = personnel.salary,  
      verify;
```

When you enter a number in the Salary field and press **RETURN**, **PERFORM** clears the field and displays

```
Please type again for verification
```

You must enter *exactly* the same data, character for character, a second time. Note that 49500 and 49500.00 are not exactly the same.

Many other attributes are available in addition to the ones discussed in this section. See Chapter 3 of the *INFORMIX-SQL Reference Manual*, “The FORMBUILD Transaction Form Generator,” for a complete list of **PERFORM** attributes.

Joining Fields

If you include data from more than one table on a form, you will probably want to include a join field. To create a join field, equate two database columns with the same field tag in the Attributes section. The following line

```
field-tag = col1 = col2;
```

specifies that **field-tag** is a join field that joins the **col1** and **col2** columns.

For example, in the **orderform** form specification file,

```
c1 = *customer.customer_num = orders.customer_num;
```

specifies that **c1** is a join field that joins the **customer.customer_num** and **orders.customer_num** columns. (The use of the asterisk in a join is described in the “Verify Join” section that follows.)

With a join field, you can assign attributes that take effect only when one table is active, as well as attributes that are always in effect. If you want an attribute to apply at all times, enter it after the last column name. For example,

```
field-tag = col1 = col2, include = (1 to 100);
```

joins **col1** to **col2** in the tag field and prevents **PERFORM** from accepting values less than 1 or greater than 100 for either **col1** or **col2**.

You can also specify an attribute that applies only when one or the other file is active. To do so, you must enter a separate attribute list for each column name. For example,

```
field-tag = col1, required, upshift;  
          = col2, upshift;
```

again joins **col1** and **col2**. When the **col1** table is active, **PERFORM** requires that you enter a value and converts that value to uppercase letters. When the **col2** table is active, you do not have to enter a value, although **PERFORM** will still convert any CHAR value you enter to uppercase letters. Additional information about the placement of attributes can be found in Chapter 3 of the *INFORMIX-SQL Reference Manual*.

The **orderform** form contains data from four of the tables in the **stores** database. Figure 6-5 shows the form specification for the **orderform** form.

database stores

screen
{

CUSTOMER INFORMATION:

Customer Number:	{ c1 }	Telephone:	{ c10 }
Company:	{ c4 }		
First Name:	{ c2 }	Last Name:	{ c3 }
Address:	{ c5 }		
	{ c6 }		
City:	{ c7 }	State:	{ c8 }
		Zip:	{ c9 }

ORDER INFORMATION:

Order Number:	{ o11 }	Order Date:	{ o12 }
Stock Number:	{ i13 }	Manufacturer:	{ i16 }
		Quantity:	{ i18 }
		Total Price:	{ i19 }

SHIPPING INFORMATION:

Customer P.O.:	{ o20 }		
Ship Date:	{ o21 }	Date Paid:	{ o22 }

}

end

tables
customer orders
items manufact

attributes

```
c1 = *customer.customer_num = orders.customer_num;  
c2 = fname,  
    comments = "Please enter initial if available ";  
c3 = lname;  
c4 = company;  
c5 = address1;  
c6 = address2;  
c7 = city;  
c8 = state, upshift, autonext,  
    include = ("CA", "OR", "NV", "WA");  
c9 = zipcode;  
c10 = phone, picture = "###-###-####x####";  
o11 = *orders.order_num = items.order_num;  
o12 = order_date,  
    default = today;  
i13 = items.stock_num;  
i16 = items.manu_code, lookup manu_name = manufact.manu_name,  
    joining *manufact.manu_code, upshift;  
i18 = quantity, include = (1 to 100);  
i19 = total_price;  
o20 = po_num;  
o21 = ship_date;  
o22 = paid_date;
```

instructions

customer master of orders;
orders master of items;

end

Figure 6-5. The orderform Form Specification

Verify Joins

Like the **INCLUDE** attribute, *verify joins* let you compare the value you enter in **PERFORM** with a list of acceptable values. With the **INCLUDE** attribute, you enter the acceptable values in the form specification. With a *verify join*, the acceptable values are stored in one of the tables in your database.

To use a verify join, you specify which column contains the acceptable values. **PERFORM** checks to see that the value you enter in a verify join field already exists in the column you specify.

To create a verify join, type an asterisk to the left of the name of the column in which the value should already exist. For example,

```
field-tag = col1 = *col2;
```

means that when the **col1** table is active, **PERFORM** rejects any input to the field that does not already exist in the **col2** column.

The **orderform** screen form in the **stores** database uses a verify join to prevent users from assigning orders to nonexistent customers:

```
c1 = *customer.customer_num = orders.customer_num;
```

When **orders** is the active table, **PERFORM** rejects any customer number you attempt to enter that does not already exist as a **customer__num** value in the **customer** table.

Lookup Joins

Lookup joins let you extract data from a table and display it on a form based on the value a user enters in one of the fields on the form. For example, the **orderform** form uses a lookup join to display the name of a manufacturer as soon as the user enters the manufacturer code.

To use a lookup join, you must create one or more fields in the Screen section for the data you want to display. These *lookup fields* differ from other fields in that **PERFORM** never displays brackets around them, and users cannot modify the data that appears in them.

You do not create a separate entry for the lookup fields in the Attributes section. Instead, include the instructions for the lookup join in the attribute list for the field on which you want **PERFORM** to base the lookup.

Format for Lookup Joins. The general format for a lookup join is

```
field-tag = col-name,  
LOOKUP lookup-tag = lookup-col JOINING join-col
```

field-tag is the field tag that appears in the Screen section for the field that **PERFORM** uses as the basis for the lookup—for example, in the **orderform** form, the manufacturer code is field **i16**.

col-name is the name of the column in which that data should be stored—again, the manufacturer-code column in the **items** table. **LOOKUP** is a required keyword.

lookup-tag is the field tag you used for the lookup field in the Screen section. The equal sign is required. In the example, **manu__name** is the lookup tag.

lookup-col is the name of the column that contains the data you want to look up. Normally, *lookup-col* is in a different table from *col-name*. JOINING is a required keyword. In the example, the **manu__name** column in the **manufact** table is the lookup column.

join-col is the column that **PERFORM** joins to *col-name* to find the data you want to look up. *join-col* is in the same table as *lookup-col*. In the example, the **manu__code** column in the **manufact** table is the join column.

An Example Lookup Join. The Screen section of the **orderform** form contains a lookup field with the field tag **manu__name**. The Attributes section includes instructions for a lookup join in the attribute list for the **i16** field:

```
i16 = items.manu_code, lookup manu_name = manufact.manu_name  
joining *manufact.manu_code, upshift;
```

When you enter a manufacturer code in the **i16** field, **PERFORM** joins the **manu__code** column in the **items** table to the **manu__code** column in the **manufact** table. Based on the join, **PERFORM** extracts the manufacturer name from the **manu__name** column in the row of the **manufact** table that matches the manufacturer code you entered.

The asterisk placed before *join-col* means that this is a verify join. **PERFORM** prevents you from entering a **manu__code** into the **items** table that does not already exist in the **manufact** table.

Instructions Section

The Instructions section of the form specification lets you take advantage of more advanced **PERFORM** and **FORMBUILD** features. Here are some of the things you can do in the Instructions section:

- Create master-detail relationships that simplify multiple-table queries.
- Perform arithmetic calculations based on the values entered in a field and display the results on a form or store them in a table.
- Use different characters to delimit the fields on a form.
- Use composite joins to join tables based on the values contained in more than one column.
- Use instruction control blocks to indicate specific processing after a user enters or changes a value in a field.

These features are discussed in detail in Chapter 3, “The **FORMBUILD** Transaction Form Generator,” in the *INFORMIX-SQL Reference Manual*.

Chapter Summary

- Creating a screen form is a two-step process. First you create a form specification, and then you compile it.
- A form specification is a system file that contains instructions describing how you want a form to look and what database data you want the form to access.
- You can use the **Generate** option on the FORM Menu to create a default form specification containing the basic instructions needed to create a form.
- You can use the default form specification as it is, or you can tailor it to your own needs.
- Every form specification must include four sections: Database, Screen, Tables, and Attributes. Some specifications also include an Instructions section.
- The Database section indicates the database used with the form.
- The Screen section shows the placement of fields and related text on the screen form.
- The Tables section lists the tables that contain data used by the form.
- The Attributes section associates each field on the form with a column in one of the tables listed in the Tables section. It can also include attributes to control the appearance and behavior of each field.
- You join fields by equating two database columns with the same field tag in the Attributes section.

- You use verify joins to prevent **PERFORM** from accepting unverified data into a table. **PERFORM** checks a second table to verify the existence of the data before it accepts the data.
- You use lookup joins to display data from one table based on input to another.

Chapter 7

Using RDSQL

Chapter 7 Table of Contents

Chapter Overview	5
Examples Used in This Chapter	5
What Is RDSQL?.....	6
An Interactive Query Language	6
Using RDSQL Statements	6
Accessing RDSQL	7
RDSQL Options	8
RDSQL SYNTAX Menu	9
Differences between RDSQL and PERFORM.....	9
How to Create a Table with RDSQL	11
Entering an RDSQL Statement	12
Using the RDSQL Editor.....	12
Using the Use-Editor Option.....	14
Naming the Table	15
Naming the Columns.....	15
Assigning Data Types.....	16
Character.....	16
Numeric.....	16
Date	17
Money	18
Running an RDSQL Statement.....	18
If There Are Errors	19
Saving the Current Statements	20
Guidelines for Assigning Command Filenames	20
The CREATE INDEX Statement.....	21
Naming the Index	21
Specifying a Table	22
Specifying Index Columns	22
Clustered Indexes.....	23
Preventing Duplicate Entries	23
Ascending and Descending Indexes	24
Using a Command File.....	25

Database Privileges on Multiuser Systems.....	26
Querying a Database	28
Entering a Statement	28
Formatting RDSQL Statements	29
Searching for Rows and Columns: The SELECT Statement	30
Searching for All Rows and All Columns.....	31
Looking at Each Row	32
Searching for Specific Columns.....	32
Arithmetic Operators.....	34
Math Functions.....	35
Date Functions	38
Searching for Specific Rows	38
Relational Operators.....	39
The MATCHES Keyword.....	41
Sorting Query Results	43
Sorting by More than One Column	45
Queries with More than One Table	46
Sending Query Results to the Printer or a File	51
Sending Results to the Printer.....	51
Sending Results to a File	52
The <i>New-File</i> Option	52
The <i>Append-file</i> Option.....	53
Modifying Data with RDSQL	54
Adding Rows to a Table.....	54
Changing the Data in a Table	57
Deleting Rows from a Table	59
Displaying Table Information	60
Chapter Summary.....	62

Chapter Overview

This chapter explains how to use **RDSQL** with the data in a database. The following topics are described:

- Creating a database with **RDSQL**
- Using **RDSQL** as an interactive query language
- Comparing **RDSQL** to **PERFORM**
- Granting database privileges to other users
- Using **RDSQL** to query a database, sort query results, and store query results
- Modify the data in a table with **RDSQL**

You should read this chapter if you want to learn how to use **RDSQL** to create tables and to work with data in databases.

For complete information on all the **RDSQL** statements, refer to Chapter 2, “The **RDSQL** Query Language,” in the *INFORMIX-SQL Reference Manual*.

Examples Used in This Chapter

Many of the **RDSQL** example statements demonstrated in this chapter are included with the demonstration database. When you list the contents of your working directory, the statements appear as **ex1.sql**, **ex2.sql**, **ex3.sql** files, and so on.

Example statements included in the database appear in the text with an example number.

What Is RDSQL?

An Interactive Query Language

RDSQL is an English-like interactive query language that you can use when working with databases. **RDSQL** is an enhanced version of SQL (Structured Query Language), the industry-standard query language developed by IBM. With **RDSQL** you can

- Create and drop tables and indexes
- Enter and delete data
- Query a database
- Select a different current database
- Display information about one or more tables
- Rename tables and columns
- Check and repair tables
- Grant and revoke database and table privileges

This chapter explains how to create tables and indexes with **RDSQL**, and how to grant database privileges. In addition, it describes how to use **RDSQL** to work with the data in a database: to retrieve, manipulate, add, remove, and modify rows and columns. Chapter 9, “Database Structure and Integrity,” considers how to use **RDSQL** to modify the structure of databases and tables.

Using RDSQL Statements

To use **RDSQL**, select the **Query-language** option on the **INFORMIX-SQL Main Menu**. The **RDSQL Menu** appears, and you can enter one or more **RDSQL statements**. A statement is simply an instruction that tells **RDSQL** what you want to do. For example, to create a table, you use the **CREATE TABLE** statement; to create an index, you use the **CREATE INDEX** statement.

You can edit the statements as you enter them using the **RDSQL** editor or a system editor. The **RDSQL** editor is discussed in detail in this chapter. The use of a system editor is noted briefly and discussed in greater detail in Chapters 6 and 8, “Creating Your Own Forms” and “Creating and Printing Reports,” respectively.

Just as there is a current database in **INFORMIX-SQL**, the **RDSQL** statements you are currently working with are called *current statements*.

RDSQL keeps a copy of current statements. You can modify and run them as many times as you like without retyping them. You can also save them in a file so you can use them again at a later time.

Accessing *RDSQL*

To access **RDSQL**, select the **Query-language** option on the **INFORMIX-SQL** Main Menu. If you have not already selected a database to work with, **INFORMIX-SQL** first displays the **CHOOSE DATABASE** screen. Once you have selected a database, **INFORMIX-SQL** displays the **RDSQL** Menu:

RDSQL:	<input type="checkbox"/>	New	Run	Modify	Use-editor	Output	Choose	Save	Info	Drop	Exit
Enter new RDSQL statements using RDSQL editor.											
..... stores Press CTRL-W for Help											

RDSQL Options

The RDSQL Menu displays 10 options:

New	Enter new RDSQL statements. The statements you enter become the current statements, replacing any previous current statements.
Run	Run the current RDSQL statements.
Modify	Modify the current RDSQL statements.
Use-editor	Use a system editor to enter or modify the current RDSQL statements.
Output	Send the results of the current RDSQL statements to a printer, file, or (on UNIX systems only) a pipe. You normally use this option only when you use RDSQL to query a database.
Choose	Choose a file that contains RDSQL statements and make those statements the current statements.
Save	Save the current RDSQL statements in a file so you can use them again later.
Info	Display information about the current database.
Drop	Drop an RDSQL command file.
Exit	Return to the INFORMIX-SQL Main Menu.

RDSQL SYNTAX Menu

In addition to the usual HELP Screens available with all INFORMIX-SQL menus and screens, the RDSQL **New**, **Modify**, and **Use-editor** options access a HELP Menu that includes information about RDSQL statement syntax and usage. Select one of those options, and then press CTRL-W. The SYNTAX Menu appears:

SYNTAX:	<input type="checkbox"/> Definition	Manipulation	Select	Privileges	Others	Help-Editor
The CREATE, DROP, DATABASE, and ALTER statements.						
..... stores Press CTRL-W for Help						

Differences between RDSQL and PERFORM

Like PERFORM, RDSQL lets you enter and modify data and query the database. With RDSQL, however, you do not use a screen form. Instead, you enter *statements*. (The “Entering an RDSQL Statement” section describes how to enter a statement.) Following is an example RDSQL statement:

```
select order_num,           {ex1}
       customer_num,
       order_date,
       paid_date
from orders
where order_date > "6/5/86"
order by paid_date
```

The statement tells RDSQL what to do. In this case, the statement tells RDSQL to select the data stored in the **order_num**, **customer_num**, **order_date**, and **paid_date** columns for every row in the **orders** table where the value in the **order_date** column is greater than 6/5/86. RDSQL sorts

the rows by the value in the **paid_date** column and displays this output on the screen. (The {ex1} characters are a comment, indicating that the statement is included in the demonstration database. Comments are ignored by **RDSQL**.)

Once you enter the statement, you can run it and look at the results. The query results look like this:

order_num	customer_num	order_date	paid_date
1006	112	09/19/1986	
1015	110	07/10/1986	08/31/1986
1013	104	09/01/1986	10/10/1986
1003	104	10/12/1986	11/04/1986
1008	110	11/17/1986	12/21/1986
1005	116	12/04/1986	12/30/1986

6 row(s) retrieved

RDSQL displays only the information you request. It does not display a complete row, for example, unless you ask for a complete row. If you want only the last names of customers with orders totaling more than \$500, that is exactly what you get—no first names, no customer numbers, no address information.

With this flexibility, you can specify which tables you want to query, join columns from different tables, perform mathematical and logical operations on data, and sort query results—all at the time that you enter the **RDSQL** query.

How to Create a Table with RDSQL

Chapter 2 describes how to create a table using the **INFORMIX-SQL** table schema editor. **RDSQL** gives you an alternate method for creating tables and databases.

You can use the **RDSQL CREATE TABLE** statement to create a table. A **CREATE TABLE** statement contains instructions that define the characteristics of a table. You must enter a **CREATE TABLE** statement (or use the table schema editor) for each table in a database.

The demonstration database contains five tables. Each was created with a different **CREATE TABLE** statement. Here is the statement that was used to create the **orders** table:

```
create table orders
(
    order_num        serial(1001),
    order_date       date,
    customer_num     integer,
    ship_instruct    char(40),
    backlog          char(1),
    po_num           char(10),
    ship_date        date,
    ship_weight      decimal(8,2),
    ship_charge      money(6),
    paid_date        date
);
```

This statement tells **RDSQL** to create a table named **orders** and specifies names and data types for ten columns. Column names appear on the left, while data types are indicated on the right.

Entering an RDSQL Statement

RDSQL statements always act on the current database. The **CREATE TABLE** statement creates a table in the current database. Before you create a table, be sure that the current database is the one in which you want the new table to appear.

When you are ready to enter the **CREATE TABLE** statement, select the **New** option on the **RDSQL** Menu. The **NEW** Screen indicates you are using the **RDSQL** editor.

NEW:	ESC = Done editing	CTRL-A = Typeover/Insert	CTRL-R = Redraw
	CTRL-X = Delete character	CTRL-D = Delete rest of line	
..... stores Press CTRL-W for Help			
<input type="checkbox"/>			

Using the RDSQL Editor

When the **NEW** Screen appears, you can start entering the **CREATE TABLE** statement using the **RDSQL** editor. The editor allows you to enter statements and edit them before you run them.

If you make a mistake, you can move the cursor with the **ARROW** keys and re-enter parts of the statement. You can also use the editing keys listed at the top of the screen to perform special functions.

Change Mode The **CTRL-A** key (or **Ins** on DOS systems) switches you back and forth between insert and typeover mode. While in insert mode, the text beneath the cursor shifts to the right as you enter new characters. While in typeover mode, characters you enter replace the

text beneath the cursor. You are automatically placed in typeover mode upon entering the **RDSQL** editor.

Redraw the Screen

The **CTRL-R** key redisplay the screen. You may use **CTRL-R** if, while entering **RDSQL** statements, you received electronic mail or some other message that makes it difficult to read what you have entered.

Delete a Character

The **CTRL-X** key (and **Del** on DOS systems) deletes the character that appears beneath the cursor.

Delete Forward

The **CTRL-D** key (and **F9** on DOS systems) deletes characters from the current cursor position to the end of the line.

When you use the **RDSQL** editor, you can enter as many lines of text as you wish. You are limited only by the memory constraints of your system, not by the size of the screen.

The **RDSQL** editor does not display more than 80 characters on a line. If you insert characters in a line so the line exceeds 80 characters, **RDSQL** displays the percent sign (%) at the end of the line to indicate that you have entered characters beyond the 80th column. Although **RDSQL** will read and compile these characters, it will be difficult for you to work with text that is not visible. It is a good idea to insert a **RETURN** somewhere in the first 80 characters of each line so that the full text will appear on the screen.

If you insert lines so that the total number of lines is greater than the screen can hold, the **RDSQL** editor scrolls down the page with the extra lines and displays the beginning and ending line numbers of the current page on the fourth line down from the top of the screen.

```
NEW:      ESC      = Done editing      CTRL-A = Typeover/Insert      CTRL-R = Redraw
          CTRL-X = Delete character    CTRL-D = Delete rest of line

-- 3 to 20 of 20 ----- stores ----- Press CTRL-W for Help -----

☐
```

If you prefer working with the system editor, you can always press **ESC** and then select the **Use-editor** option on the **RDSQL** Menu.

Using the Use-Editor Option

For a very long series of statements, you may wish to use a system editor. When you know that the statements you will enter are long, you can choose the **Use-editor** option right away, instead of entering some text with the **New** option and then changing to the **Use-editor** option.

If this is the first time in this session that you have used the **Use-editor** option, **RDSQL** may display the **USE-EDITOR** Screen:

```
USE-EDITOR >>vi☐
Enter editor name. (RETURN only for default editor)

----- stores ----- Press CTRL-W for Help -----
```

On this screen, **vi** is the default editor. (You may have a different default editor on your system.) You can specify your choice of editor with the **DBEDIT** environment variable. (See Appendix B for instructions.) If you have already specified an editor with this screen or if you have specified an editor with **DBEDIT**, **RDSQL** calls up the editor immediately and does not display the **USE-EDITOR** Screen.

When the USE-EDITOR Screen appears, type the name of the editor you want to use, or press **RETURN** to select the default editor. **RDSQL** calls the editor you specified. Edit the text and then save the file, following the usual rules for the editor you select. The **RDSQL** Menu is redisplayed.

Naming the Table

The **CREATE TABLE** statement begins by assigning a name to the table you are creating. You can choose any name you want, but you must follow the guidelines for naming tables described in Chapter 2, “Creating a Database.” For example,

```
create table mytable
```

is the beginning of a **CREATE TABLE** statement that creates a table named **mytable**.

Naming the Columns

After the table name, enter a name and data type for each column in the table. Be sure to enclose this information in parentheses. Here is the general format:

```
CREATE TABLE table-name  
  (column-name  data-type, . . .  
   column-name  data-type,  
   column-name  data-type)
```

The rules for assigning column names are the same as those that apply to table names. These guidelines were discussed in detail in Chapter 2. Here is a quick review:

- Column names can be up to eighteen characters long.
- Column names must be unique within the table.
- Names must begin with a letter and may contain only letters, numbers, and underscores.
- You cannot use any **INFORMIX-SQL** reserved words. (See Appendix C for a list of reserved words.)

Assigning Data Types

You must assign a data type to each column in a table. The data type indicates the kind of information you intend to store in each column. Data types were considered in Chapter 2. A detailed discussion of data types appears in Chapter 9, “Database Structure and Integrity.” Here is a brief description of each data type, with examples.

Character

CHAR columns store any combination of letters, numbers and symbols. You must specify the length for each CHAR column and enclose the length in parentheses. For example,

```
create table mytable (mycolumn char(15))
```

creates a table named **mytable** containing a character column named **mycolumn** that holds up to 15 characters.

Numeric

Six different numeric data types are available to store different kinds of numeric data. Chapter 9 notes the characteristics of DECIMAL, SMALLINT, INTEGER, SMALLFLOAT, FLOAT, and SERIAL data types.

Here is an example of a CREATE TABLE statement with each numeric data type:

```
create table mytable           {ex2}
  (col1    decimal(6,2),
   col2    smallint,
   col3    integer,
   col4    smallfloat,
   col5    float,
   col6    serial(101)
  )
```

An interpretation of the CREATE TABLE statement follows:

- col1** A DECIMAL column with fixed precision (six places) and scale (two places to the right of the decimal point).
- col2** A SMALLINT column that only stores whole numbers.
- col3** An INTEGER column that only stores whole numbers.
- col4** A SMALLFLOAT column that stores single-precision, floating-point numbers.
- col5** A FLOAT column that stores double-precision, floating-point numbers.
- col6** A SERIAL column. **INFORMIX-SQL** automatically assigns a number, beginning with 101, to the column as you add rows to the table.

Date

The DATE data type stores calendar dates. For example,

```
create table mytable (mydate date)
```

creates a table named **mytable** containing a DATE column named **mydate**.

Money

MONEY columns store currency amounts. For example,

```
create table mytable (mymoney money(8,2))
```

creates a table named **mytable** containing the MONEY column **mymoney** with fixed precision (eight places) and scale (two places to the right of the decimal point).

Running an RDSQL Statement

After you enter the CREATE TABLE statement, press **ESC** to tell **RDSQL** you are finished. When you press **ESC**, you will see the **RDSQL** Menu again; the statement you entered will appear in the bottom half of the screen.

```
RDSQL: ☐ New ☒ Run ☐ Modify Use-editor Output Choose Save Info Drop Exit
Run the current RDSQL statements.

----- stores ----- Press CTRL-W for Help -----

create table mytable
(col1      decimal(6,2),
 col2      smallint,
 col3      integer,
 col4      smallfloat,
 col5      float,
 col6      serial(101),
 mydate    date,
 mymoney   money(8,2)
)
```

To run the statement and create the table, select **Run** on the **RDSQL** Menu. The **Run** option is already highlighted, so just press **RETURN**.

If There Are Errors

When you select the **Run** option, **RDSQL** first checks the statements to make sure they conform to the **RDSQL** grammar and syntax rules. If your statements contain no mistakes, **RDSQL** processes the statements and displays the results on the screen.

If you make any errors in a statement, **RDSQL** does not process the statement. Instead, it displays the statements you entered, along with a message describing the error. For example, if there is a mistake (perhaps a misspelled keyword), **RDSQL** displays an error message on the bottom of the screen and highlights the **Modify** option in the **RDSQL** Menu.

```
RDSQL: ☐ New Run ☒ Modify Use-editor Output Choose Save Info Drop Exit
Modify the current RDSQL statements using the RDSQL editor.

----- stores ----- Press CTRL-W for Help -----

creat table mytable
(col1      decimal(6,2),
col2      smallint,
col3      integer,
col4      smallfloat,
col5      float,
col6      serial(101),
mydate    date,
mymoney   money(8,2)
)

201: A syntax error has occurred.
```

When you select **Modify**, **RDSQL** calls the **RDSQL** editor and positions the cursor on the first error. You can correct the mistake using the **RDSQL** editor (press **ESC** when you are finished editing the statement), or you can select the **Use-editor** option and edit the statement using your system editor. Once you return to the **RDSQL** Menu you can run the statement again.

Note: The text of all error messages, along with suggestions for corrections, is included in a special section at the back of the *INFORMIX-SQL Reference Manual*.

Saving the Current Statements

When you have entered and run the CREATE TABLE statement successfully, you can save it in a *command file*. A command file is a system file that contains one or more RDSQL statements.

To save the current statements in a command file, select the **Save** option on the RDSQL Menu. **INFORMIX-SQL** displays the **SAVE** Screen and prompts you to enter a name for the command file.

```
SAVE >>☐
Enter the name you want to assign to the command file.

----- stores ----- Press CTRL-W for Help -----

create table mytable
  (col1      decimal(6,2),
   col2      smallint,
   col3      integer,
   col4      smallfloat,
   col5      float,
   col6      serial(101),
   mydate    date,
   mymoney   money(8,2)
  )
```

Guidelines for Assigning Command Filenames

Command filenames can be up to eight (in DOS) or ten (in UNIX) characters long. The first character must be a letter, but you can use letters, numbers, and underscores () for the rest of the name. You can use uppercase and lowercase letters. UNIX systems are case sensitive; **ords1** is not the same as **Ords1** or **ORDS1**. DOS systems are not case sensitive.

Enter a name for the command file that will hold your statements and press **RETURN**.

RDSQL stores the statements in a command file, using the name you gave and the extension **.sql**. For example, a statement you called **ex1** is stored in the command file **ex1.sql**. You can inspect the statements at any time with the **Choose** option on the RDSQL Menu.

The CREATE INDEX Statement

You use the **RDSQL CREATE INDEX** statement to create indexes for the columns in a table. You need to enter a **CREATE INDEX** statement for each index you want to create.

The **customer__num** column in the **orders** table is indexed because it is used to join other tables in the demonstration database. Here is the statement used to create the index for the **customer__num** column.

```
create index o__c__num__ix on orders (customer__num)
```

This statement tells **RDSQL** to create an index named **o__c__num__ix** for the **customer__num** column in the **orders** table. The following sections explain this statement in detail.

A detailed discussion of indexes and indexing strategies is contained in Chapter 9, “Database Structure and Integrity.”

Naming the Index

You must assign a name to each index that you create. The **CREATE INDEX** statement begins by assigning the index name. For example,

```
create index o__c__num__ix
```

is the beginning of a **CREATE INDEX** statement that creates an index named **o__c__num__ix**. You may want to choose

names for indexes that are similar to the names of the columns you index; this will make it easier to remember the index names.

You follow the same guidelines in assigning index names as you do with table names:

- Index names may be up to eighteen characters long, and must be unique throughout a database.
- The first character of the name must be a letter; the rest of the name may consist of letters, numbers, and underscores.
- **INFORMIX-SQL** makes no distinction between uppercase and lowercase letters.
- You cannot use an **INFORMIX-SQL** reserved word as a name.

Specifying a Table

After the index name, enter the word **on** followed by the name of the table in which the column or columns you are indexing can be found. For example,

```
create index o__c__num__ix on orders
```

indicates that the index column is located in the **orders** table.

Specifying Index Columns

Finally, you need to list the name of the column or columns you want to index after the table name. For example,

```
create index o__c__num__ix on orders (customer__num)
```

indicates that you want the **o__c__num__ix** index to apply to the **customer__num** column.

To create a multiple-column index, enter a series of column names separated by commas. For example,

```
create index myindex on mytable (mycol1, mycol2)
```

creates an index named **myindex** for the **mycol1** and **mycol2** columns in the table named **mytable**. For example,

```
create index st__man__ix on items (stock__num, manu__code)
```

creates an index named **st__man__ix** for the **stock__num** and **manu__code** columns in the **items** table.

Clustered Indexes

Since both UNIX and DOS extract information from the disk in blocks, the more rows that are physically on the same block and are already in the same order as an index, the faster an indexed retrieval proceeds. You can cause the physical order in the table to be the same as the order in an index through *clustering*. A more detailed discussion of clustered indexes can be found in Chapter 2 of the *INFORMIX-SQL Reference Manual*.

Preventing Duplicate Entries

If you wish to prevent users from entering duplicate information in an indexed column, enter **unique** in the **CREATE INDEX** statement.

For example, to prevent users from entering duplicate values in the **order__num** column of the **orders** table, use the following **CREATE INDEX** statement:

```
create unique index o__num__ix on orders (order__num)
```

This statement creates an index named **o__num__ix** that applies to the **order__num** column in the **orders** table and prevents users from entering duplicate values.

Ascending and Descending Indexes

If the user does not specify the sorting order, **RDSQL** creates an index in ascending order—that is, from *A* to *Z* for **CHAR** fields, from low to high for numeric and **MONEY** fields, and from earlier to later in time for **DATE** fields. You can also create descending indexes with the **CREATE INDEX** statement.

Create descending indexes only for fields you intend to sort in descending order on a regular basis. For example, create a descending index for a date field if you normally sort the data in a table in reverse chronological order.

To create a descending index, enter **desc** after the column name in the **CREATE INDEX** statement.

```
create index myind on mytable (mycol desc)
```

This statement creates a descending index named **myind** that applies to the **mycol** column in the table named **mytable**.

Using a Command File

When you save statements in a command file, you can use them again at any time. To do so, select the **Choose** option on the RDSQL Menu. The CHOOSE Screen then appears and displays a list of the names of command files to which you have access. In addition to the command files shown in the following screen, you will see other command files that have been added to provide further practice once you are familiar with the demonstration database.

```
CHOOSE >>☐
Choose a command file with the Arrow Keys, or enter a name, then press Return.

----- stores ----- Press CTRL-W for Help -----

c_custom      ex11      ex2
c_index       ex12      ex3
c_items       ex13      ex4
c_manuf       ex14      ex5
c_orders      ex15      ex6
c_stock       ex16      ex7
c_stores      ex17      ex8
ex1           ex18      ex9
ex10          ex19
```

To select a command file, use the **ARROW** keys to move the highlight over its name, then press **RETURN**. You can also type the name of the file you want and press **RETURN**.

After selecting a command file, the RDSQL Menu is displayed. The statements contained in the command file are now the current statements, and they appear on the screen. You can modify or run these statements.

Database Privileges on Multiuser Systems

The person who creates a database is considered the Database Administrator (DBA) and is the only one who has access to the database. If other users are to have access to the database, the DBA must grant these users database privileges. Until privileges are extended to other users, only the DBA can create and drop tables and indexes (to be discussed in Chapter 9), add and retrieve table data, and grant database privileges.

INFORMIX-SQL uses two levels of privileges to ensure database security and data integrity. Database privileges control access to the database and the ability to create tables in the database. Table privileges control access to individual tables and the ability to create, alter, query, and update each table. Separate privilege statements are used to grant the appropriate access level to users.

Three database privileges are available:

CONNECT	Allows a user to access a database.
RESOURCE	Allows a user to create or drop tables and indexes in a database.
DBA	Allows a user all the privileges of the DBA, including the ability to alter the system tables, to drop, start, and rollforward the database, and to grant CONNECT , RESOURCE , and DBA privileges to others. START DATABASE and ROLLFORWARD DATABASE statements are described in Chapter 2 of the <i>INFORMIX-SQL Reference Manual</i> .

The most commonly granted database privilege is the **CONNECT** privilege. Without **CONNECT** privilege, a user cannot query the database and add or update table data.

You can grant any privileges to **PUBLIC**, rather than to specified users, if you want all users to have a privilege.

The general format of a database GRANT statement is

```
GRANT privilege TO {PUBLIC | user-list}
```

To grant CONNECT privileges to PUBLIC, you would run this statement:

```
grant connect to public
```

All GRANT statements operate on the current database.

A complete list of **INFORMIX-SQL** database and table privileges is contained in Chapter 2, “The RDSQL Query Language,” of the *INFORMIX-SQL Reference Manual*.

Querying a Database

The preceding discussion described how to use the RDSQL Menu to create a table. A summary of the procedures for querying the database with **RDSQL** follows.

The **stores** database is the current database for the remaining portion of this chapter. If it is not your current database, enter the following **RDSQL** statement:

database stores

Entering a Statement

Select the **New** option on the RDSQL Menu and type the **RDSQL** statement shown on the screen. The top of the screen shows the editing keys you can use. When you finish entering the statement, press **ESC**.

```
NEW:  ESC      = Done editing      CTRL-A = Typeover/Insert    CTRL-R = Redraw
      CTRL-X = Delete character    CTRL-D = Delete rest of line
----- stores ----- Press CTRL-W for Help -----

select order_num, customer_num,      |ex3|
      paid_date
      from orders
      order by paid_date
```

The RDSQL Menu again appears, with the **Run** option highlighted. Press **RETURN** or type **r** to run your statement.

```
RDSQL: ☐ New ☒ Run ☐ Modify Use-editor Output Save Choose Info Exit
Run the current RDSQL statements.

----- stores ----- Press CTRL-W for Help -----

select order_num, customer_num,      |ex3|
      paid_date
      from orders
      order by paid_date
```

Formatting RDSQL Statements

You can use almost any format you like for **RDSQL** statements. You can put every word on a different line or type as much as you can fit on each line. This query

```
select                                {ex4}
    customer_num,
    lname,
    city,
    phone
from customer
```

is the same as this query

```
select customer_num, lname, city, phone
      from customer                                {ex4}
```

The sample statements in this chapter are formatted so you can read them easily. You can include comments in your **RDSQL** statements; just enclose your comments in { } braces like this:

```
select                                {beginning of statement}
    customer_num,                    {list of columns}
    lname,
    city,
    phone
from customer                        {table in stores db}
```

Comments are for your use only; **RDSQL** does not process the comments.

You can include several **RDSQL** statements at one time, but you must separate them with semicolons. For example,

```
select order_num from orders;      {ex5}
select customer_num, lname,
       city, phone
from customer
```

Semicolons are statement separators, not terminators. This means you do not have to type a semicolon at the end of your last statement. **RDSQL** sequentially processes your statements and displays the output.

Searching for Rows and Columns: The **SELECT** Statement

Use the **SELECT** statement to query a database and display the results on the screen. The **SELECT** statement is the basis of all queries in **RDSQL**. You can use seven clauses in a **SELECT** statement:

- The **SELECT** clause (required)
- The **FROM** clause (required)
- The **WHERE** clause
- The **ORDER BY** clause
- The **GROUP BY** clause
- The **HAVING** clause
- The **INTO TEMP** clause

The **SELECT** and **FROM** clauses are required; the others are optional. In this chapter, you will learn how to use the first four clauses in this list; see Chapter 2 of the *INFORMIX-SQL Reference Manual* for information on the other clauses.

Searching for All Rows and All Columns

You can use **RDSQL** to look at one, several, or all rows in a table. Unless you specify otherwise, **RDSQL** finds all rows. The general format for finding all rows and all columns in a table is

```
SELECT * FROM tablename
```

Here are the rules:

- Use the **SELECT** keyword followed by an asterisk to tell **RDSQL** that you want to find *all* the columns in the table.
- Use the **FROM** keyword followed by a table name to tell **RDSQL** which table contains the data you are searching for.

To look at all the rows in the **items** table, for example, type

```
select * from items
```

and press **ESC**. Choose the **Run** option to execute the **SELECT** statement. The **RUN** Menu shown below is displayed.

RUN: <input type="checkbox"/> <input checked="" type="checkbox"/> Next Restart Exit					
Display next page of query results.					
----- stores ----- Press CTRL-W for Help -----					
item_num	order_num	stock_num	manu_code	quantity	total_price
1	1001	1 HRO	1	\$250.00	
1	1002	4 HSK	1	\$960.00	
2	1002	3 HSK	1	\$240.00	
1	1003	9 ANZ	1	\$20.00	
2	1003	8 ANZ	1	\$840.00	
3	1003	5 ANZ	5	\$99.00	
1	1004	1 HRO	1	\$960.00	
2	1004	2 HRO	1	\$126.00	
3	1004	3 HSK	1	\$240.00	
4	1004	1 HSK	1	\$800.00	
1	1005	5 NRG	10	\$280.00	
2	1005	5 ANZ	10	\$198.00	
3	1005	6 SMT	1	\$36.00	
4	1005	6 ANZ	1	\$48.00	

Looking at Each Row

RDSQL displays as many rows as it can fit on the screen. The rows can be arranged horizontally or vertically, depending on their size. If all the retrieved rows fit on one screen “page,” they appear under the **RDSQL Menu**.

If the results do not fit on one page (as in the example above), **RDSQL** displays them under the **RUN Menu** so that you can look through the pages with the **Next** option. Use **Next** to display the next page of results. When you reach the last row, **RDSQL** displays a message on the bottom of the screen showing how many rows it found. In this case, the message reads

39 row(s) retrieved.

To look at the query results again, select the **Restart** option to go back to the first screen page. You can use the **Next** option to step through the following screen pages.

When you are finished looking at the query results, use the **Exit** option on the **RUN Menu** to return to the **RDSQL Menu**.

Searching for Specific Columns

You can also search for data from specific columns in a table. Instead of an asterisk, list the names of the columns you want, separated by commas. The general format is

```
SELECT col1 [, col2, ... coln] FROM table-name
```

Here are the **RDSQL** rules for retrieving specific columns:

- Follow the **SELECT** keyword with a list of column names, separated by commas.
- List the columns in the order you want **RDSQL** to display them.

- Follow the FROM keyword with the name of the table that contains the columns.

For example, if you want to see only the customer number, last name, city, and phone number of all the customers listed in the **customer** table, enter:

```
select customer_num, lname,
       city, phone
from customer
```

Press **ESC** and then select the **Run** option. Here are the results:

RUN: <input type="checkbox"/> Next <input type="checkbox"/> Restart <input type="checkbox"/> Exit Display next page of query results.			
----- stores ----- Press CTRL-W for Help -----			
customer_num	lname	city	phone
101	Pauli	Sunnyvale	408-789-8075
102	Sadler	San Francisco	415-822-1289
103	Currie	Palo Alto	415-328-4543
104	Higgins	Redwood City	415-368-1100
105	Vector	Los Altos	415-776-3249
106	Watson	Mountain View	415-389-8789
107	Ream	Palo Alto	415-356-9876
108	Quinn	Redwood City	415-544-8729
109	Miller	Sunnyvale	408-723-8789
110	Jaeger	Redwood City	415-743-3611
111	Keyes	Sunnyvale	408-277-7245
112	Lawson	Los Altos	415-887-7235
113	Beatty	Menlo Park	415-356-9982
114	Albertson	Redwood City	415-886-6677

RDSQL shows fourteen rows because that is what fits on the first screen page. The other four rows and the message

18 row(s) retrieved.

appear on the next screen page.

Arithmetic Operators

You can use arithmetic operators with any numeric column in a **SELECT** clause. The symbols you use for each function are as follows:

Symbol	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division

The following statement calculates the unit price of each item in stock if the price of each item is increased by five percent:

```
select stock_num, description, {ex6}  
      unit, unit_descr, unit_price,  
      unit_price * 1.05  
from stock
```

The expression `unit_price * 1.05` calculates the new price. Here are the results:

RUN: <input type="checkbox"/> <input checked="" type="checkbox"/> Next Restart Exit					
Display next page of query results.					
----- stores ----- Press CTRL-W for Help -----					
stock_num	description	unit	unit_descr	unit_price	(expression)
1	baseball gloves	case	10 gloves/case	\$250.00	\$262.5000
1	baseball gloves	case	10 gloves/case	\$800.00	\$840.0000
1	baseball gloves	case	10 gloves/case	\$450.00	\$472.5000
2	baseball	case	24/case	\$126.00	\$132.3000
3	baseball bat	case	12/case	\$240.00	\$252.0000
4	football	case	24/case	\$960.00	\$1008.0000
4	football	case	24/case	\$480.00	\$504.0000
5	tennis racquet	each	each	\$28.00	\$29.4000
5	tennis racquet	each	each	\$25.00	\$26.2500
5	tennis racquet	each	each	\$19.80	\$20.7900
6	tennis ball	case	24 cans/case	\$36.00	\$37.8000
6	tennis ball	case	24 cans/case	\$48.00	\$50.4000
7	basketball	case	24/case	\$600.00	\$630.0000
8	volleyball	case	24/case	\$840.00	\$882.0000

The heading for the fifth column is **(expression)**. This heading is provided by **RDSQL** for columns that do not exist in the

AVG	Finds the average value in the column you specify.
MAX	Finds the maximum (highest) value in the column you specify.
MIN	Finds the minimum (lowest) value in the column you specify.

When using the SUM, AVG, MAX, and MIN functions with a column, you enclose the column name in parentheses. If you include more than one function in a SELECT clause, separate the functions with commas. Do not specify a column name with the COUNT(*) function.

For example, you could find the average, minimum, and maximum shipping charge for orders listed in the **orders** table with this SELECT statement:

```
select avg(ship_charge),           {ex8}
       min(ship_charge),
       max(ship_charge)
from orders
```

The results would look like this:

RDSQL: <input type="checkbox"/> New <input checked="" type="checkbox"/> Run <input type="checkbox"/> Modify Use-editor Output Choose Save Info Drop Exit			
Run the current RDSQL statements.			
----- stores ----- Press CTRL-W for Help -----			
(avg)	(min)	(max)	
\$13.13	\$5.00	\$25.20	

Each aggregate finds one result for the whole table or for a group of rows instead of one result for each row. You cannot put aggregates in a SELECT statement when you want to see individual rows.

Aggregates are very useful with the GROUP BY clause, as explained in Chapter 2 of the *INFORMIX-SQL Reference Manual*.

INFORMIX-SQL automatically attaches the headings (**avg**), (**min**), and (**max**) to the columns. You can give the columns a different heading by typing a display label after the calculations in the **SELECT** statement that produce the column. The following statement changes each heading:

```
select avg(ship_charge)  avg_charge,    {ex9}
       min(ship_charge) min_charge,
       max(ship_charge) max_charge
from orders
```

The results look like this:

RDSQL: <input type="checkbox"/> New <input checked="" type="checkbox"/> Run <input type="checkbox"/> Modify Use-editor Output Choose Save Info Drop Exit			
Run the current RDSQL statements.			
----- stores ----- Press CTRL-W for Help -----			
avg_charge	min_charge	max_charge	
\$13.13	\$5.00	\$25.20	

NULL values affect the aggregate functions in the following ways:

- All rows returned with a **SELECT COUNT (*)** statement are counted, including **NULL** rows.
- **NULL** rows are ignored by the **SUM**, **AVG**, **MAX**, and **MIN** aggregate functions.

Date Functions

You can use the built-in date functions anywhere in an expression that a quoted string or a number may appear. When you use these functions with a column, enclose the column name in parentheses and separate the functions with commas.

DATE	Returns a DATE value corresponding to the expression with which you call it.
DAY	Returns the day of the month.
MDY	Returns a DATE value when you call it with three expressions that evaluate to integers representing the month, day, and year.
MONTH	Returns the month of the year.
WEEKDAY	Returns the day of the week.
YEAR	Returns the year.

Display labels can also be used with date functions.

The uses of date functions are described in detail in Chapter 2 of the *INFORMIX-SQL Reference Manual*.

Searching for Specific Rows

To search for some, but not all, rows in a table, include a WHERE clause in the SELECT statement. You use the WHERE clause to specify the search criteria **RDSQL** uses to determine which rows to retrieve. The general format is

WHERE *conditions*

The search *conditions* describe the acceptable values for one or more columns. In this chapter, you will learn about using relational operators and the keyword **MATCHES** to form search conditions.

Relational Operators

The **RDSQL** query language uses six *relational operators*. They are called relational operators because they describe a relationship between two values. You can use any of the following relational operators in a **WHERE** clause:

Symbol	Meaning
=	Equal to
<> or !=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

The simplest format for using relational operators in a **WHERE** clause is

WHERE *column relational-operator value*

The effect of a relational operator on **DATE** or **CHAR** data types may not be immediately apparent.

- With dates, greater than means later in time, and less than means earlier in time.
- With characters, greater than means later in the alphabet, and less than means earlier in the alphabet.
- Lowercase letters are greater than uppercase letters, and uppercase letters are greater than numbers.

RDSQL compares characters based on the ASCII character set. (Appendix F of the *INFORMIX-SQL Reference Manual* contains an ASCII chart.)

Here are the rules for using relational operators in a **SELECT** statement with a **WHERE** clause:

- Follow the **WHERE** keyword with search conditions or descriptions of the rows you want to find.

- When search conditions include a column name, a relational operator, and a value, enclose CHAR and DATE values in quotation marks.

For example, suppose you want to search for all rows in the **orders** table that were shipped on or before 06/10/86. You would use a WHERE clause to eliminate rows for those orders shipped after 06/10/86:

```
select order_num, order_date,    {ex10}
       ship_date, customer_num
from orders
where
  ship_date <= "06/10/86"
order by order_num
```

This is what you will see:

```
RDSQL: ☐ New ☒ Run ☐ Modify Use-editor Output Choose Save Info Drop Exit
Run the current RDSQL statements.

----- stores ----- Press CTRL-W for Help -----

order_num order_date ship_date customer_num

1001 01/20/1986 02/01/1986 104
1002 06/01/1986 06/06/1986 101
1004 04/12/1986 04/30/1986 106
1007 03/25/1986 04/23/1986 117
1009 02/14/1986 03/04/1986 111
1010 05/29/1986 06/08/1986 115
1011 03/23/1986 04/13/1986 104
1012 06/05/1986 06/09/1986 117
1014 05/01/1986 05/10/1986 106

9 row(s) retrieved.
```


You can use more than one search condition in a WHERE clause, connecting the search conditions with the AND keyword. You can also include alternative search conditions with the OR keyword. The following SELECT statement searches for rows in the stock table where the unit price falls between \$100 and \$400 and the stock number is less than 3.

```
select * from stock where      {ex11}
    (unit_price > 100.00 and
     unit_price < 400.00)
    and stock_num < 3
```

Two rows in the table match these conditions:

RDSQL: <input type="checkbox"/> New <input checked="" type="checkbox"/> Run <input type="checkbox"/> Modify <input type="checkbox"/> Use-editor <input type="checkbox"/> Output <input type="checkbox"/> Choose <input type="checkbox"/> Save <input type="checkbox"/> Info <input type="checkbox"/> Drop <input type="checkbox"/> Exit									
Run the current RDSQL statements.									
----- stores ----- Press CTRL-W for Help -----									
stock_num	manu_code	description		unit_price	unit	unit_descr			
1	HRO	baseball gloves		\$250.00	case 10	gloves/case			
2	HRO	baseball		\$126.00	case 24	case			

The MATCHES Keyword

The MATCHES keyword lets you find rows containing a particular word or phrase. The general format is

WHERE *char__column* MATCHES "*value*"

Here are the rules for using the MATCHES keyword in a SELECT statement with a WHERE clause:

- Follow the WHERE keyword with search conditions or descriptions of the rows you want to find.
- When search conditions include a column name, the MATCHES keyword, and a value, the column must be a CHAR column and the value must be enclosed in quotation marks.

The strength of the MATCHES keyword lies in the three *wildcard* characters you can use with it:

Symbol	Meaning
--------	---------

- | | |
|-------|--|
| * | Matches zero or more characters |
| ? | Matches any single character |
| [...] | Matches any one of a specified string of characters.
The string can be implied by way of a range. |

Examples of the use of the MATCHES keyword with wildcard characters follow. Note that the wildcard character is included within the quotes that surround the selection criteria:

- | | |
|-------------------|--|
| MATCHES "S*" | Finds any character column that begins with an uppercase <i>S</i> . |
| MATCHES "**ere" | Finds any character column that ends with the letters <i>ere</i> . |
| MATCHES "**an*" | Finds any character column that includes the letters <i>an</i> . |
| MATCHES "an?" | Finds any character column containing three letters that begins with <i>an</i> . |
| MATCHES "?an" | Finds any character column containing three letters that ends in <i>an</i> . |
| MATCHES "r?t" | Finds any character column containing three letters that begins with <i>r</i> and ends with <i>t</i> . |
| MATCHES "[A-R]at" | Finds any character column containing three letters that begins with the letters <i>A</i> through <i>R</i> and ends with <i>at</i> . |

For example, to find information about customers whose names end in *son*, enter,

```
select customer_num,      {ex12}
       lname, city, phone
from customer
       where lname matches "*son"
```

RDSQL retrieves these rows:

RDSQL: <input type="checkbox"/> New <input checked="" type="button" value="Run"/> Modify Use-editor Output Choose Save Info Drop Exit			
Run the current RDSQL statements.			
----- stores ----- Press CTRL-W for Help -----			
customer_num	lname	city	phone
106	Watson	Mountain View	415-389-8789
112	Lawson	Los Altos	415-887-7235
114	Albertson	Redwood City	415-886-6677

The * wildcard character substitutes for all characters that occur before the *son* in each name. The NOT option (NOT MATCHES) makes the search condition true when the column on the left does NOT match the specified pattern. If NOT appeared before MATCHES in the preceding example, all names *not* ending in *son* would be retrieved.

Sorting Query Results

You can use the ORDER BY clause to sort query results by any column specified in the SELECT clause. The general format is

```
ORDER BY col1 [, col2,...coln]
```

Here are the rules for using the ORDER BY clause in a SELECT statement:

- Follow the ORDER BY keywords with the names of the columns you want to sort by.

- Use only columns included in the SELECT clause in the ORDER BY clause.

For example, the following query sorts stock information by unit price:

```
select stock_num, description,      {ex13}
       unit, unit_descr, unit_price
from stock
order by unit_price
```

Here is the first page of the results:

RUN: <input type="checkbox"/> Next <input type="checkbox"/> Restart <input type="checkbox"/> Exit Display next page of query results.				
..... stores Press CTRL-W for Help				
stock_num	description	unit	unit_descr	unit_price
5	tennis racquet	each	each	\$19.80
9	volleyball net	each	each	\$20.00
5	tennis racquet	each	each	\$25.00
5	tennis racquet	each	each	\$28.00
6	tennis ball	case	24 cans/case	\$36.00
6	tennis ball	case	24 cans/case	\$48.00
2	baseball	case	24/case	\$126.00
3	baseball bat	case	12/case	\$240.00
1	baseball gloves	case	10 gloves/case	\$250.00
1	baseball gloves	case	10 gloves/case	\$450.00
4	football	case	24/case	\$480.00
7	basketball	case	24/case	\$600.00
1	baseball gloves	case	10 gloves/case	\$800.00
8	volleyball	case	24/case	\$840.00

You can only use columns named in the SELECT clause to order the rows. In the prior example you could order the rows by any column named in the SELECT clause: **stock_num**, **description**, **unit**, **unit_price**, or **unit_descr**.

Sorting by More than One Column

You can also use the ORDER BY clause for *nested sorts*, or sorts within sorts. When you use more than one sort column, separate the column names following the ORDER BY keywords with commas.

Customer information can be ordered by city and last name with the following SELECT statement:

```
select customer_num, lname,      {ex14}
       city, phone
from customer
order by city, lname
```

This SELECT statement produces the following results:

RDSQL: <input type="checkbox"/> New <input checked="" type="checkbox"/> Run <input type="checkbox"/> Modify Use-editor Output Choose Save Info Drop Exit			
Run the current RDSQL statements.			
----- stores ----- Press CTRL-W for Help -----			
customer_num	lname	city	phone
112	Lawson	Los Altos	415-887-7235
105	Vector	Los Altos	415-776-3249
113	Beatty	Menlo Park	415-356-9982
115	Grant	Menlo Park	415-356-1123
116	Parmelee	Mountain View	415-534-8822
106	Watson	Mountain View	415-389-8789
118	Baxter	Oakland	415-655-0011
103	Currie	Palo Alto	415-328-4543
107	Ream	Palo Alto	415-356-9876
114	Albertson	Redwood City	415-886-6677
104	Higgins	Redwood City	415-368-1100
110	Jaeger	Redwood City	415-743-3611
108	Quinn	Redwood City	415-544-8729
117	Sipes	Redwood City	415-245-4578

The rows are first sorted by **city**. When there is more than one row in a **city** category, the rows in that category are sorted by **lname**.

Queries with More than One Table

You can retrieve data from more than one table in a query, as long as all the tables you are working with belong to the same database. The general format is

```
SELECT [table1.]col1 [, [table2.]col2,...coln]
FROM table1, table2 [...tablen]
```

Here are the rules for using more than one table in a query:

- Follow the **SELECT** keyword with the names of the columns you want to find, arranged as you want to see them in the results and separated with commas.
- Follow the **FROM** keyword with the names of the tables you want to use, arranged in any order and separated with commas.
- When columns from two or more tables have the same name, preface the column name with the table name. Use a period (.) to separate the table name and the column name.

If you are using join columns (columns that appear in more than one table), one additional rule applies:

- List join columns only once in the **SELECT** clause, unless you want the same value to print twice.

For example, the **stock** table contains information about the sporting goods stock of the store, and the **manufact** table contains information about each manufacturer that ships goods to the store. The **stock** table has these columns:

```
stock__num
manu__code
description
unit__price
unit
unit__descr
```

The **manufact** table has these columns:

```
manu__code  
manu__name
```

Both tables contain **manu__code** columns.

By joining the column that the two tables have in common, you can query both tables at the same time and find information about stock and the manufacturers of each item.

The following **SELECT** statement searches the database for the stock number, manufacturer code and name, item description, unit price, and unit price increased by five percent for all stock items beginning with the word *baseball* in their description. It finds all but the manufacturer name in the **stock** table. The **manu__code** column in the **stock** table is joined to the **manu__code** column in the **manufact** table.

```
select stock_num, stock.manu_code,      {ex15}  
       manu_name, description, unit_price,  
       unit_price * 1.05 newprice  
from stock, manufact  
where stock.manu_code  
= manufact.manu_code and  
description matches "baseball*"  
order by stock.manu_code
```

You can list either the **stock.manu__code** or the **manufact.manu__code** column in the **SELECT** clause; it makes no difference which you use. However, if you are going to sort by the **manu__code** column, you must use the **manu__code** column listed in the **SELECT** clause in the **ORDER BY** clause. You cannot list a column in one table (the **stock.manu__code** column) in the **SELECT** clause and a column in a second table (the **manufact.manu__code** column) in the **ORDER BY** clause.

Here are the results:

```

RDSQL: ☐ New  Modify Use-editor Output Choose Save Info Drop Exit
Run the current RDSQL statements.

----- stores ----- Press CTRL-W for Help -----

stock_num manu_code manu_name      description      unit_price      newprice
1  HRO      Hero      baseball gloves  $250.00         $262.5000
2  HRO      Hero      baseball         $126.00         $132.3000
1  HSK      Husky     baseball gloves  $800.00         $840.0000
3  HSK      Husky     baseball bat     $240.00         $252.0000
1  SMT      Smith     baseball gloves  $450.00         $472.5000

5 row(s) retrieved.

```

A query involving three tables follows the same rules. The **items** table contains the following information about the items the store sells:

item_num
order_num
stock_num
manu_code
quantity
total_price

It can be linked to the **stock** table by the **stock_num** and **manu_code** columns; both tables share this information.

The following query retrieves information from all three tables. It finds the manufacturer's code and name, unit, description, unit price, quantity, total price, and order number for all orders of more than one of the same item. It sorts the results by order number.

```
select stock.manu_code, manu_name,      {ex16}
       unit, description, unit_price,
       quantity, total_price, order_num
from stock, manufact, items
where
  stock.manu_code = manufact.manu_code
and
  stock.manu_code = items.manu_code
and
  stock.stock_num = items.stock_num
and
  quantity > 1
order by order_num
```

Here is a portion of the results:

manu_code	ANZ
manu_name	Anza
unit	each
description	tennis racquet
unit_price	\$19.80
quantity	5
total_price	\$99.00
order_num	1003
manu_code	NRG
manu_name	Norge
unit	each
description	tennis racquet
unit_price	\$28.00
quantity	10
total_price	\$280.00
order_num	1005
manu_code	ANZ
manu_name	Anza
unit	each
description	tennis racquet
unit_price	\$19.80
quantity	10
total_price	\$198.00
order_num	1005

The results of the query are presented as a row output because it is not possible to fit all the columns in the display area.

Multiple-table queries are one of the most powerful uses of **INFORMIX-SQL**. Chapter 2 of the *INFORMIX-SQL Reference Manual* describes them in greater detail.

Sending Query Results to the Printer or a File

The output from an **RDSQL** statement normally appears on the screen. The **Output** option on the **RDSQL** Menu lets you send query results to the printer, store them in a system file, or (on UNIX systems) send them to a pipe. This section discusses the first two operations only. For more information on the **Output** option, see Chapter 2 in the *INFORMIX-SQL Reference Manual*.

When you use the **Output** option, the query is printed or stored just as it appears on the screen. (Format control and other sophisticated report features available for use with query results are explained in Chapter 8, “Creating and Printing Reports.”)

On UNIX systems, the **OUTPUT** Menu looks like this:

OUTPUT:	<input type="checkbox"/>	Printer	New-file	Append-file	To-pipe	Exit
Send query results to a printer.						
----- stores ----- Press CTRL-W for Help -----						

On DOS systems, the **To-pipe** option is not available.

Sending Results to the Printer

If you want to send your query results directly to the printer, select the **Printer** option on the **OUTPUT** Menu. **RDSQL** sends the query results directly to your printer and displays a message on the bottom of the screen telling how many rows it retrieved. The query results do not appear on the screen.

Sending Results to a File

INFORMIX-SQL lets you write query results to a new file or append the results to an existing file. You can use your operating system programs to edit, copy, rename, or delete the file.

The New-File Option

If you want to create a new file to store query results, select the **New-file** option. You will see the **OUTPUT NEW-FILE** Screen.

```
OUTPUT NEW-FILE >>☐  
Enter the name you want to assign to the new file, then press Return.  
..... stores ..... Press CTRL-W for Help .....
```

Type a name for the file, and then press the **RETURN** key. **RDSQL** outputs the results of the query into the file and displays a message telling how many rows were retrieved. The query results do not appear on the screen.

Caution! If you enter the name of an existing file, **RDSQL** overwrites the existing file with the query results.

The *Append-file* Option

If you want to add your query results to the end of an existing file without replacing what is already stored there, select the **Append-file** option. You will see the OUTPUT APPEND-FILE Screen.

```
OUTPUT APPEND-FILE >>☐
```

```
Enter the name of the file you want to append results to, then press Return.
```

```
..... stores ..... Press CTRL-W for Help .....
```

Type the name of the file you want to append the query results to, and press the RETURN key. RDSQL outputs the query results to the file and displays a message telling how many rows were retrieved. The query results do not appear on the screen.

Modifying Data with RDSQL

You can use **RDSQL** to add, modify, and delete rows in a database using the **INSERT**, **UPDATE**, and **DELETE** statements.

Adding Rows to a Table

You add data to a table by listing each column with the values you want to add in the **INSERT** statement. The general format is

```
INSERT INTO table  
  [ (col1 [,col2,...coln) ]  
VALUES (val1 [,val2,...valn])
```

For part of a row, you must list the relevant columns and values. You do not need to include the column list for a complete row. Columns that are not included in the list are assigned a **NULL** value.

Here are the rules for adding a row with **RDSQL**:

- Follow the **INSERT INTO** keywords with the name of the table.
- Enclose the column list in parentheses, and separate each name with commas.
- Non-serial columns that do not allow **NULL** values must be included in the column list and receive values.
- Follow the **VALUES** keyword with the data you want to add.
- Enclose the data in parentheses; type the values in the same order as the column names, and separate them with commas.
- Put quotation marks around each **CHAR** and **DATE** data item.

- Use the keyword **NULL** to insert a **NULL** value into a column.
- The number of values specified in the **VALUES** clause must be equal to the number of columns in the table or to the number of columns in the optional column list.

For example, suppose you want to add this information about a new customer to the **customer** table:

customer__num	
fname	Richard
lname	Sitruc
company	Tennis World
address1	322 El Camino Real
address2	
city	Burlingame
state	CA
zipcode	94250
phone	415-322-4100

This is what you type:

```
insert into customer          {ex17}
values
(0, "Richard", "Sitruc",
 "Tennis World", "322 El Camino Real",
 null, "Burlingame", "CA", "94250",
 "415-322-4100")
```

After you type the statement, press **ESC**, and then select the **Run** option. **RDSQL** displays this message:

1 row(s) inserted.

Now you can select that row with a command like the one below, and inspect it.

```
select *
  from customer
 where lname matches "Sitruc"
```

RDSQL displays:

```
RDSQL: ☐ New ☒ Run ☐ Modify ☐ Use-editor ☐ Output ☐ Choose ☐ Save ☐ Info ☐ Drop ☐ Exit
Run the current RDSQL statements.

----- stores ----- Press CTRL-W for Help -----

customer_num    119
fname           Richard
lname           Sitruc
company         Tennis World
address1        322 El Camino Real
address2
city            Burlingame
state           CA
zipcode         94250
phone           415-322-4100

1 row(s) retrieved
```

You should not specify a value for a SERIAL column in the INSERT statement. Because each entry for a SERIAL column must be unique, it is easier to simply enter a zero (0) as a place holder for the SERIAL value and let RDSQL add the proper value. In the prior statement, INFORMIX-SQL automatically assigned the next available customer number (119) to the row when the customer was added to the database.

The INSERT statement is most useful for adding a group of rows to a table. Instead of specifying a value for each column with the VALUES clause, you can use a SELECT clause to choose and modify data from one table and add it to another table. See the INSERT statement in Chapter 2 of the *INFORMIX-SQL Reference Manual* for an example.

If you add more than a few rows, it is a good idea to follow the INSERT statement with the UPDATE STATISTICS statement. See Chapter 2 of the *INFORMIX-SQL Reference Manual* for details about the UPDATE STATISTICS statement.

Changing the Data in a Table

You modify the data in a table by setting one or more columns equal to values specified in the UPDATE statement. You can use a WHERE clause to specify which row you want to change. The general format for an UPDATE statement is

```
UPDATE table-name SET {column-name = expr [, ... ] |  
  {(column-list)*} = (expr-list)}  
  [WHERE (iicondition)]
```

Here are the rules for changing data with RDSQL:

- Follow the UPDATE keyword with the name of the table.
- Follow the SET keyword with the name of a column, an equal sign, and the value you want to enter for that column.
- Enclose CHAR and DATE data in quotation marks.
- Specify a value of NULL for columns that accept NULL values.
- Use a WHERE clause to select the row or rows you want to update.
- If you want to change data in more than one column, you can list all column-names receiving new values in a column-list, followed by a list of the values. This is demonstrated by the following:

```
UPDATE customer  
  SET (fname, company, address2) =  
      ("Marie", "Marie's Sports", "P. O. Box 3621"  
  WHERE customer_num = 103
```

You can enter one column name or a series of column names, separated by commas. You can enter one value or a series of values, separated by commas. The number of items in the column-list must equal the number of items in the value-list.

You can also use the asterisk to UPDATE the values of all columns in a row:

```
UPDATE stock SET * =  
    (5, "NRG", "tennis racquet", "295.00", "case",  
     "10/case")  
WHERE stock_num = 5 and manu_code = "NRG"
```

The asterisk is expanded to reference all columns in the table.

Caution! If you leave out the WHERE clause, **RDSQL** assumes you want to change *all* rows. In interactive mode, **RDSQL** does not run the UPDATE statement until you confirm that you want to change all rows. However, when the UPDATE statement is in a command file and you are running **RDSQL** from the operating system command line, **RDSQL** executes it immediately.

See Appendix M for information about running **RDSQL** from the command line.

For example, the zipcode column was incorrectly entered for the row of customer number 119. The following UPDATE statement is used to correct the situation:

```
update customer  
    set zipcode = "94205",  
    where customer_num = 119
```

After you type the statement, press **ESC**, and then select the **Run** option. **RDSQL** displays

1 row(s) updated.

You can use a **SELECT** statement to display the updated row.

The **UPDATE** statement is most useful for changing groups of rows. You could add five percent to all prices simply by executing the following statement:

```
update stock
set unit_price = unit_price * 1.05
```

Deleting Rows from a Table

You delete a row from a table by using a **WHERE** clause to specify the row that you want to delete. You cannot delete part of a row; use the **UPDATE** statement for replacing the data in a column with a **NULL** or zero. The general format for a **DELETE** statement is

DELETE FROM *table* **WHERE** clause

Here are the rules for deleting a row with **RDSQL**:

- Follow the **DELETE FROM** keywords with the name of the table you want to delete a row or rows from.
- Use a **WHERE** clause to specify the row or rows you want to delete.

Caution! If you leave out the **WHERE** clause, **RDSQL** assumes you want to delete *all* rows. In interactive mode, **RDSQL** does not run the **DELETE** statement until you confirm that you want to remove all rows. However, when a **DELETE** statement without a **WHERE** clause is part of a command file and you are running **RDSQL** from the operating system command line, **RDSQL** executes it immediately.

See Appendix M for information about running **RDSQL** from the command line.

To delete the data for Richard Sitruc, type this statement:

```
delete from customer
  where fname = "Richard"
     and lname = "Sitruc"
```

After you type the statement, press **ESC**, and then select the **Run** option. **RDSQL** displays

```
1 row(s) deleted.
```

Richard Sitruc's row is no longer in the database. If you delete more than a few rows, it is a good idea to follow the **DELETE** statement with the **UPDATE STATISTICS** statement. See Chapter 2 in the *INFORMIX-SQL Reference Manual* for details.

Displaying Table Information

Use the **Info** option on the **RDSQL** Menu to display information about the columns, indexes, privileges, and status of a table.

After you select the **Info** option, **RDSQL** displays the **INFO FOR TABLE** Screen with a list of tables. Type or highlight the name of the table that you want and press **RETURN**. The **INFO** Menu then appears.

INFO - customer:	<input type="checkbox"/> Columns	Indexes	Privileges	Status	Table	Exit
Display column names and data types for a table.						
..... stores Press CTRL-W for Help						

Six options are available on the **Info** Menu:

Columns	Lists all the columns in the table and the data type of each column.
Indexes	Lists the indexes for the tables, the owner and type of each index, and the columns to which each index applies.
Privileges	Lists the users who have access privileges for the table, and the RDSQL data statements (SELECT, UPDATE, INSERT, DELETE, INDEX, and ALTER) that can be used with the data in the table. (public is a general category for all users; unless your login is listed separately, you have the privileges given for public .)
Status	Lists the table name, the table owner, the size of the row (in number of characters), the number of rows in the table (as of the last UPDATE STATISTICS statement), the number of columns in a row, the date the table was created, and the name of the audit trail file, if there is one.
Table	Selects a new table for examination with the INFO Menu.
Exit	Returns to the RDSQL Menu.

The **Info** Menu is also available from the TABLE Menu.

Chapter Summary

- **RDSQL** is an interactive query language that allows you to create databases and find, add, change, and remove data in a database.
- You use the **RDSQL Menu** to enter the statements to tell **RDSQL** what to do.
- You use the **SELECT** statement in all queries. You can specify the columns you want to find in the **SELECT** clause, the tables you want to work with in the **FROM** clause, the rows you want to find with the **WHERE** clause, and the order of query results with the **ORDER BY** clause.
- You can query more than one table at a time when you specify more than one table in the **FROM** clause and join columns containing the same information with the **WHERE** clause.
- You can put a query in a command file with the **Save** option and send query results to the printer or a file with the **Output** option.
- You can add rows to a table with the **INSERT** statement.
- You can change the contents of rows in a table with the **UPDATE** statement.
- You can remove rows from a table with the **DELETE** statement.

Chapter 8

Creating and Printing Reports

Chapter 8 Table of Contents

Chapter Overview	5
What Is a Report Writer?.....	6
Steps in Creating a Report.....	7
Using the REPORT Menu.....	8
A Sample Specification and Report.....	8
Sections of a Specification	11
The DATABASE Section	11
The SELECT Section	11
The FORMAT Section.....	12
Creating a Report Specification.....	14
Generating a Default Report Specification	15
Naming the Report	15
Compiling the Default Report Specification.....	16
Running the Report.....	17
Modifying a Report	20
Saving the Report Specification.....	21
Compiling a Report Specification	23
If the Compilation Is Unsuccessful.....	23
What a Report Specification Contains.....	26
Listing the Instructions.....	26
Including Comments	26
Selecting the Data for a Report	28
SELECT and FORMAT Work Together.....	28
Selecting All Rows and Columns of a Table.....	29
Selecting Data from More than One Table	30
Basic Report Formatting	31
The OUTPUT Section	31
The FORMAT Section.....	32
FORMAT for a Default Report.....	32

FORMAT for a Custom Report	33
Creating Page Headers and Trailers.....	33
Where Headers and Trailers Appear	33
Printing a Report Heading	34
Printing Column Headings	35
Numbering Pages Automatically	36
Printing Rows of Data.....	37
Organizing the Data.....	39
Grouping Data.....	40
The Group Control Blocks.....	41
Calculations in a Report	43
Arithmetic Operators.....	43
Example of Arithmetic Operations	44
Using Parentheses.....	45
Controlling the Appearance of Numbers.....	45
Using Fill Characters	46
The Fill Characters.....	46
Adding a Dollar Sign	47
Making the Dollar Sign Float	48
Math Functions.....	48
Totaling a Column	49
Counting Rows in a Report	51
Combining Arithmetic Operators and Math Functions	53
Group Functions	54
Displaying a Report.....	57
Printing the Report on a Printer	57
Writing the Report to a File.....	58
Piping the Report to Another Program	59
Chapter Summary.....	60

Chapter Overview

This chapter explains how to create, display, and print reports with **INFORMIX-SQL**. The following topics are considered:

- Using a report writer
- Creating and compiling a report specification
- Including selected database information in a report
- Grouping and formatting the information
- Adding titles, headings, and other text to customize the report
- Displaying the report on your terminal or sending it to the printer

Read this chapter if you want to learn how to create your own **INFORMIX-SQL** reports. If you intend only to use reports that someone else creates, you do not need to read this entire chapter. Instead, go on to the section “Running the Report.”

For complete information about all the features available for creating and running reports, see Chapter 5, “The ACE Report Writer,” in the *INFORMIX-SQL Reference Manual*.

What Is a Report Writer?

A report consists of database information, arranged and formatted according to your instructions, and displayed on the screen, printed on paper, or stored in a file for future use. You can select all or some of the information stored in a database and display it in a report.

INFORMIX-SQL includes a *report writer* that you use to select and format the information you want to include in your reports. The report writer is called **ACE**.

The **ACE** report writer retrieves data from a database, organizes and formats it according to your instructions, adds the headings or column titles you specify, performs calculations on numeric columns, and then displays the report.

You can combine information from several tables in a database to create whatever kind of report you need. Some of the kinds of reports **ACE** can produce are:

- Mailing labels
- Form letters
- Payroll summaries
- Medical histories
- Accounting records
- Inventory lists
- Telephone directories
- Project schedules
- Sales and marketing reports
- Payroll checks

Example report specifications are included in Appendix E of the *INFORMIX-SQL Reference Manual*. You may find them useful models for creating similar reports for your databases.

Steps in Creating a Report

To create and display a report in **INFORMIX-SQL**, you need to complete the following three tasks:

1. Create a *report specification*. A report specification is a list of instructions that tells **INFORMIX-SQL** what information you want to include in the report, how you want it organized and grouped, what headings or titles you want to add, what calculations you want to perform, and what margins and other formatting elements you want to use.
2. Compile the report specification. During compilation, **INFORMIX-SQL** checks to see if the specification is correct and complete. If there are any errors, you must correct them and recompile the specification.
3. Run the compiled report. When you run the report, **INFORMIX-SQL** reads the database, formats the information, performs the requested operations, and either displays the report on your screen, prints it on the printer, or sends the report to a file.

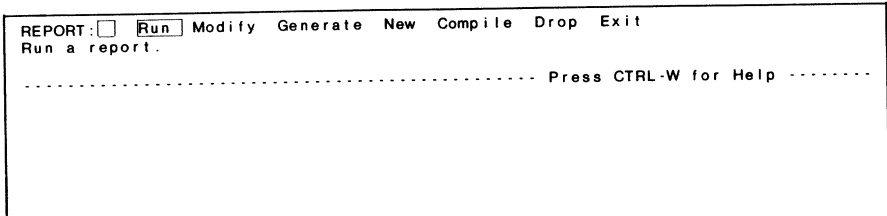
Figure 8-1 shows an overview of the steps involved in creating and displaying a report.

Step 1.	Create a report specification.
Step 2.	Compile the specification.
Step 3.	If there are errors, correct the errors, and then go back to Step 2.
Step 4.	When there are no more compilation errors, run the report.

Figure 8-1. How to Create a Report

Using the REPORT Menu

The **Report** option available from the Main Menu leads to the menus used to create, modify, compile, and run reports.



A screenshot of a terminal window showing the REPORT menu. The menu is titled 'REPORT:' and has a list of options: Run, Modify, Generate, New, Compile, Drop, and Exit. The 'Run' option is highlighted with a cursor. Below the menu, there is a prompt 'Run a report.' and a dashed line with the text 'Press CTRL-W for Help'.

The **Report** Menu displays seven options:

Run	Displays a report.
Modify	Changes an existing report specification.
Generate	Generates a default report specification.
New	Creates a new report specification.
Compile	Compiles a report specification.
Drop	Drops a report.
Exit	Returns to the INFORMIX-SQL Main Menu.

A Sample Specification and Report

Figure 8-2 shows an example report you can produce from the **stores** database.

CUSTOMER LIST

September 30, 1986

Last Name	Company	City
Vector Lawson	Los Altos Sports Runners & Others	Los Altos Los Altos
Beatty Grant	Sportstown Gold Medal Sports	Menlo Park Menlo Park
Watson Parmelee	Watson & Son Olympic City	Mountain View Mountain View
Baxter	Blue Ribbon Sports	Oakland
Currie Ream	Phil's Sports Athletic Supplies	Palo Alto Palo Alto
Higgins Quinn Jaeger Albertson Sipes	Play Ball! Quinn's Sports AA Athletics Sporting Place Kids Korner	Redwood City Redwood City Redwood City Redwood City Redwood City
Sadler	Sports Spot	San Francisco
Pauli Miller Keyes	All Sports Supplies Sport Stuff Sports Center	Sunnyvale Sunnyvale Sunnyvale

Total Number of Customers: 18

Confidential Information

Page 1

Figure 8-2. A Sample Report

This report lists the last name, company, and city of all customers. It groups customers by city, and calculates the total number of customers. The words "Confidential Information" and the current page number are printed at the bottom of each page.

Figure 8-3 shows the report specification used to generate the sample report.

```
database
    stores
end

select
    lname, company, city
    from customer
    order by city
end

format

page header
    print column 20, "CUSTOMER LIST"
    skip 1 line
    print column 18, "September 30, 1986"
    skip 2 lines
    print "Last Name",
    column 18, "Company",
    column 40, "City"
    skip 2 lines

on every row
    print lname,
    column 18, company,
    column 40, city

after group of city
    skip 2 lines

on last row
    skip 2 lines
    print column 12, "Total Number of Customers:",
    column 40, count using "##"

page trailer
    print "Confidential Information",
    column 55, pageno using "Page ##"

end
```

Figure 8-3. A Sample Report Specification

Sections of a Specification

The sample report specification in Figure 8-3 contains three sections: DATABASE, SELECT, and FORMAT. All report specifications must contain at least these three sections.

Following is a brief explanation of what each section in the sample report specification contains.

The DATABASE Section

All report specifications must begin with a DATABASE section. The DATABASE section tells **INFORMIX-SQL** which database contains the information you want to use in the report.

```
database
  stores
end
```

In the sample specification, the information for the report is contained in the **stores** database.

The SELECT Section

The SELECT section of a report specification contains one or more SELECT statements. The SELECT statements work exactly the same way in report specifications as they do in **RDSQL**.

You use a **SELECT** statement to specify which rows and columns contain the information you want to include in the report.

```
select
    lname, company, city
from customer
order by city
end
```

In the example, the **SELECT** statement contains three clauses.

SELECT	Select the information in the last name, company, and city columns.
FROM	The columns are contained in the customer table.
ORDER BY	Order the rows in ascending order by city (city).

For a more complete explanation of the **SELECT** statement, refer to Chapter 7, “Using RDSQL,” in this guide and Chapter 2, “The RDSQL Query Language,” in the *INFORMIX-SQL Reference Manual*.

The **FORMAT** Section

The **FORMAT** section controls the appearance of a report. You can add report titles and column headings, group information in a specific way, perform calculations, and add a footer at the bottom of the report.

```
format

page header
    print column 20, "CUSTOMER LIST"
    skip 1 line
    print column 18, "September 30, 1986"
    skip 2 lines
    print "Last Name",
    column 18, "Company",
    column 40, "City"
    skip 2 lines

on every row
    print lname,
    column 18, company,
    column 40, city

after group of city
    skip 2 lines

on last row
    skip 2 lines
    print column 12, "Total Number of Customers:",
    column 40, count using "##"

page trailer
    print "Confidential Information",
    column 55, pageno using "Page ##"

end
```

In this example, the FORMAT section contains a list of instructions in the following five control blocks:

PAGE HEADER	Print a report title ("CUSTOMER LIST"), skip one line, and print a date ("September 30, 1986"). Skip two lines and print column headings ("Name," "Company," and "City") in the designated column positions. Skip two lines.
ON EVERY ROW	Print the contents of the last name, company, and city columns of every row in the designated column positions.

AFTER GROUP OF	Skip two lines following the printing of each group of rows clustered by city.
ON LAST ROW	After selecting the last row, skip two lines, and print a phrase (“Total Number of Customers”) in the designated column position. Calculate and display the total number of customers.
PAGE TRAILER	At the bottom of each page print the words “Confidential Information” and the current page number.

Formatting control blocks are described in detail later in this chapter.

Creating a Report Specification

Before you can create your report, you have to tell **INFORMIX-SQL** which database you will be working with and give the name of the report you want to create.

To begin creating a report specification, select the **Report** option from the Main Menu. **INFORMIX-SQL** displays the **REPORT** Menu:

```

REPORT: ☐ Run ☐ Modify ☐ Generate ☐ New ☐ Compile ☐ Drop ☐ Exit
Run a report.
..... stores ..... Press CTRL-W for Help .....

```

The **REPORT** Menu gives you two ways to create a report specification. You can select the **Generate** option and generate a default report specification, or select the **New** option and use your system text editor to create a new report

specification. The next few pages describe the process of generating a default report specification. The use of the **New** option is covered in Chapter 5 of the *INFORMIX-SQL Reference Manual*.

Generating a Default Report Specification

To create a default report specification, select the **Generate** option. If you have not yet selected the database with which you want to work, **INFORMIX-SQL** displays the **CHOOSE DATABASE** Screen. Type or highlight the name of the database you want to work with (select **stores** so you can work with the demonstration database) and press **RETURN**. If you have already selected a database, **INFORMIX-SQL** proceeds immediately to the **GENERATE REPORT** Screen.

```
GENERATE REPORT >>[ ]
Enter the name you want to assign to the report, then press Return.

..... stores ..... Press CTRL-W for Help .....
```

Once you select the **Generate** option (and choose a database), **INFORMIX-SQL** asks you to enter the name you want to assign to the new report specification.

Naming the Report

You can assign any name you want to the report specification under the following conditions:

- The name can be up to 8 (in DOS) or 10 (in UNIX) characters long.
- The report specification name must begin with a letter and can consist of any combination of letters, numbers, and

underscores (_). You can use uppercase and lowercase letters. UNIX systems recognize case; DOS systems make no distinction between uppercase and lowercase letters.

After you enter the name of the report specification, INFORMIX-SQL displays the CHOOSE TABLE Screen. Below are displayed some of the available tables.

```
CHOOSE TABLE >>  
Choose the table to be used in the default report.  
----- stores ----- Press CTRL-W for Help -----  
  
  
  
items  
  
manufact  
  
orders  
  
stock
```

You now choose the table upon which INFORMIX-SQL will base the default report. The default report will display information drawn from this table. Select a table from those listed on the screen (select **customer** to work with the **customer** table included with the demonstration database).

Compiling the Default Report Specification

A report specification must first be compiled before INFORMIX-SQL can run the report. Compiling allows INFORMIX-SQL to convert the specification into a special format that is used by ACE.

After selecting a table for the default report specification, **INFORMIX-SQL** compiles the specification, saves the specification file, and returns you to the **REPORT** Menu. **ACE** can now use the compiled report specification.

The default report specification does not take advantage of many of the powerful features available with **ACE**. For example, the default report only includes one table, the data display may be unattractive, and the report does not perform any data manipulation. You will probably want to make changes to the report specification file before you use it. The remaining sections of this chapter describe ways in which you can tailor a report to meet your specific needs.

Running the Report

When you run a report, **INFORMIX-SQL** reads the compiled report specification, locates the specified database, and selects and formats the information according to the instructions in the report specification.

To run a report, select the **Run** option on the **REPORT** Menu:

```
REPORT: ☐ ☒ Run Modify Generate New Compile Drop Exit
Run a report.
----- stores ----- Press CTRL-W for Help -----
```

INFORMIX-SQL displays the **RUN REPORT** Screen with a list of available reports (reports in addition to those listed here are available for further practice):

```
RUN REPORT >>☐
Choose a report with Arrow Keys, or enter a name, then press Return.
----- stores ----- Press CTRL-W for Help -----

clist1
clist2
mail1
mail2
mail3
ord1
ord2
ord3
```

Type or highlight the name of the report you want to run and press **RETURN**. **INFORMIX-SQL** runs the selected report.

Depending on the instructions that the report specification contains, the finished report is either displayed on your screen, written to a file, or printed on your printer.

By default, **INFORMIX-SQL** displays the results on your screen. Subsequent sections of this chapter describe how to store a report in a file or print it on the system printer.

The first 32 lines of the default report based on the **customer** table appear as follows:

customer_num	101
fname	Ludwig
lname	Pauli
company	All Sports Supplies
address1	213 Erstwild Court
address2	
city	Sunnyvale
state	CA
zipcode	94086
phone	408-789-8075

customer_num	102
fname	Carole
lname	Sadler
company	Sports Spot
address1	785 Geary St
address2	
city	San Francisco
state	CA
zipcode	94117
phone	415-822-1289

customer_num	103
fname	Philip
lname	Currie
company	Phil's Sports
address1	654 Poplar
address2	P. O. Box 3498
city	Palo Alto
state	CA
zipcode	94303
phone	415-328-4543

Modifying a Report

If you decide to make changes to an existing report, you need to modify the report specification and then recompile it.

To change a report specification, select the **Modify** option on the REPORT Menu:

```
REPORT: ☐ Run ☒ Modify ☐ Generate ☐ New ☐ Compile ☐ Drop ☐ Exit
Modify a report specification.

----- stores ----- Press CTRL-W for Help -----
```

INFORMIX-SQL displays the MODIFY REPORT Screen with a list of available report specifications (reports in addition to those listed here are available for further practice):

```
MODIFY REPORT >>☐
Choose a report with Arrow Keys, or enter a name, then press Return.

----- stores ----- Press CTRL-W for Help -----

clist1
clist2
mail1
mail2
mail3
ord1
ord2
ord3
```

Type or highlight the name of the report specification you want to modify and press **RETURN**.

After selecting the specification, **INFORMIX-SQL** asks for the name of your editor (unless you have specified an editor previously in this session or set the **DBEDIT** environment variable as described in Appendix B). **INFORMIX-SQL** then places you in your system text editor with the report specification file. You can make any changes you like to the specification, using your system editor.

When finished entering changes to the specification, save the file according to the procedures you normally follow to save files with your text editor.

Saving the Report Specification

After saving the file, you will see the **MODIFY REPORT Menu**:

MODIFY REPORT:	<input type="checkbox"/>	Compile	Save-and-exit	Discard-and-exit
Compile the report specification.				
----- stores ----- Press CTRL-W for Help -----				

Three options are available on the **MODIFY REPORT Menu**:

Compile

Compiles the report specification and returns to the **MODIFY REPORT Menu** if the compilation was successful. Once the report specification is compiled, it can be saved and run.

Save-and-exit

Saves the report specification without compiling and returns to the **REPORT Menu**. The report specification file is available for editing and/or compiling at a later time.

Discard-and-exit Discards the modifications just made to the report specification and returns to the REPORT Menu.

Normally, you will want to compile the report specification immediately. To do so, select the **Compile** option.

Compiling a Report Specification

When you compile a report specification, **INFORMIX-SQL** reads each line of the specification to see if you entered everything completely and correctly. If the specification is correct, **INFORMIX-SQL** proceeds with the compilation. When the compilation is completed, **INFORMIX-SQL** displays a message to that effect and the **MODIFY REPORT** Menu reappears. You can then select the **Save-and-exit** option and run the report.

If the Compilation Is Unsuccessful

If the report specification contains instructions that **INFORMIX-SQL** cannot read or understand, it stops compiling and displays an error message on the screen. You need to correct the errors if **INFORMIX-SQL** is to successfully compile the specification.

The following is a portion of a report specification with an error. The **DATABASE** section is missing the **END** keyword.

```
database
  stores
                                {end keyword is missing}

select
  lname, company, city
  .
  .
  .
```

Figure 8-4. A Report Specification Containing an Error

After attempting to compile a report specification containing an error, **INFORMIX-SQL** displays a message indicating that it found an error and displays the **COMPILE REPORT** Menu.

COMPILE REPORT: <input type="checkbox"/> <input checked="" type="checkbox"/> Correct <input type="checkbox"/> Exit		
Correct errors in the report specification.		
..... stores Press CTRL-W for Help		

To correct an error in the specification, select the **Correct** option from the **COMPILE REPORT** Menu. When you do, **INFORMIX-SQL** calls your system editor. The report specification appears on your screen, with errors flagged by a caret (^) and an error message.

Note: The text of all error messages, along with suggestions for corrections, is included in a special section at the back of the *INFORMIX-SQL Reference Manual*.

Figure 8-5 shows what the erroneous specification file would look like after you attempted to compile it.

```
database
  stores
                                [end keyword is missing]

select
#^
#
#   A grammatical error has been found on line 4, character 1.
# The construct is not understandable in its context.
# See error number -8018.
#
  lname, company, city
```

Figure 8-5. A Report Specification with an Error Marked

In the example shown in Figure 8-5, you need to correct the report specification by adding the keyword **END** to the **DATABASE** section.

Suggestion: In the example, ACE flagged the word “select” as the point at which the error was encountered. In fact it is the lack of the END keyword that caused the error. This example demonstrates a common occurrence with ACE: An error is often located in the control block immediately preceding the flag. When you encounter a grammatical error, look to the block prior to where the flag is pointing and check for a missing or misspelled keyword, or a missing or misplaced punctuation mark.

After correcting the error, save the specification file in the usual way and exit from your text editor. **INFORMIX-SQL** removes the error message from the specification file when you save it. Choose the **Compile** option again to recompile the specification.

If the report compiles successfully, **INFORMIX-SQL** displays the MODIFY REPORT Menu. Choose the **Save-and-exit** option. **INFORMIX-SQL** saves the compiled report specification and displays the REPORT Menu.

What a Report Specification Contains

A report specification is divided into sections. Each section begins with the section name (such as **DATABASE**, **SELECT**, or **FORMAT**), followed by control blocks (such as **ON EVERY ROW** or **ON LAST ROW**) and statements (such as **PRINT** or **SKIP**), and concludes with the keyword **END**. The sections must appear in the sequence noted below. A report must include the three sections marked as required:

- **DATABASE** (required)
- **DEFINE**
- **INPUT**
- **OUTPUT**
- **SELECT** (required)
- **FORMAT** (required)

The control blocks contain statements, and most of the statements can contain clauses. For a complete description of an ACE report specification, refer to the overview section of Chapter 5, “The ACE Report Writer,” in the *INFORMIX-SQL Reference Manual*.

Listing the Instructions

Within a section you can list the control blocks and statements in any style or format you like, as long as you enter them correctly. **INFORMIX-SQL** ignores extra spaces and **RETURNS** in the specification.

Including Comments

You can include comments in your report specification. That way, if you need to modify the report or if someone else reads the specification, there are comments explaining what the specification does.

Include comments within a set of { } braces. Text enclosed in braces is ignored by **INFORMIX-SQL**. For example, the following report specification includes comments.

```

{
This report lists customers by company and
city and groups by city
}
database                               {select stores database}
    stores
end

output                                {set left margin to 10}
    left margin 10
end

select                                {select report information}
    lname, company, city
    from customer
    order by city                      {order by city}
end

format

page header                            {set up page headings}
    print column 20, "CUSTOMER LIST"
    skip 1 line
    print column 18, "September 30, 1986"
    skip 2 lines
    print "Last Name",
    column 18, "Company",
    column 40, "City"
    skip 2 lines

on every row                            {format every row of output}
    print lname,
    column 18, company,
    column 40, city

after group of city
    skip 2 lines                        {skip 2 lines after each group}

on last row                             {count number of rows}
    skip 2 lines
    print column 12, "Total Number of Customers:",
    column 40, count using "##"

page trailer                            {at bottom of page}
    print "Confidential Information",
    column 55, pageno using "Page ##"

end

```

Figure 8-6. A Report Specification with Comments

Selecting the Data for a Report

The next part of this chapter explains how you can create your own report based on your database. Examples using data from the **stores** database illustrate the actions of the various format commands.

The **SELECT** statement is used to specify what information you want to include in your report. Use the same clauses for the **SELECT** statement that you use in **RDSQL**:

- **SELECT** clause (required)
- **FROM** clause (required)
- **WHERE** clause
- **GROUP BY** clause
- **HAVING** clause
- **ORDER BY** clause
- **INTO TEMP** clause

Refer to Chapter 7, “Using **RDSQL**,” in this guide, and Chapter 2, “The **RDSQL** Query Language,” in the *INFORMIX-SQL Reference Manual* for more information about the **SELECT** statement.

SELECT and FORMAT Work Together

You list the columns you want **INFORMIX-SQL** to retrieve from the database in the **SELECT** section. However, you need not display each column in the report. For example, you might include one of the columns because you want to use it in an **ORDER BY** clause; or perhaps you want to use one of the columns in a calculation. You do not necessarily want to print it in the report.

To select but not print columns in the report, list all the column names in the **SELECT** section. Then, list only those columns you want to print in the **FORMAT** section.

For example, to select customers' last names, store names, and customer numbers from the **customer** table and order them by customer number (**customer_num**), use a SELECT statement like this:

```
select customer_num,    {selecting data}
       company, lname
       from customer
       order by customer_num
end
```

To print only the last names and store names, enter the FORMAT section like this:

```
format
  on every row
  print          {printing data}
  lname, company
end
```

Selecting All Rows and Columns of a Table

If you want to select all the rows and columns of a table, you can use an asterisk (*) with the SELECT statement. For example, to select all the rows and columns of the **orders** table, you would enter

```
select *              {select all rows from orders table}
  from orders
end
```

Selecting Data from More than One Table

RDSQL syntax is used to select information in a report. For reports drawing upon information found in more than one table, include a WHERE clause in the SELECT statement. The WHERE clause indicates which tables contain the columns you want to join.

Following is a SELECT statement that joins the **stock** table and the **manufact** table:

```
select stock_num, stock.manu_code,
       manu_name, description, unit,
       unit_price
  from stock, manufact
 where
       stock.manu_code =
       manufact.manu_code
end
```

By joining the two tables (on the **manu-code** column) you can include information from the columns in both tables in your report. This example selects stock numbers, manufacturer codes, stock descriptions, units, and unit prices from the **stock** table, and manufacturer names from the **manufact** table.

Basic Report Formatting

You can format the report to suit your reporting needs. You can create columnar reports, as well as reports with irregular formats, such as payroll checks or mailing labels. The **FORMAT** and **OUTPUT** sections of your report specification control the appearance and contents of a report. In these sections, you can

- Create page headers, page trailers, and column headings
- Print rows of information in the body of the report
- Group information
- Perform calculations, such as totals and averages
- Set up page margins and dimensions
- Specify whether you want the report to go to the screen, a file, the printer, or (on UNIX systems) a pipe.

The OUTPUT Section

You can use the **OUTPUT** section to set left and right margins and the page length for your report. In addition, you can use the **OUTPUT** section to send the report to a printer, write the report to a file, or (on UNIX systems) pipe the report output as input to another program.

You do not need to include an **OUTPUT** section in every report specification. Without an **OUTPUT** section, **INFORMIX-SQL** uses default settings, and the report is displayed on the screen.

The default output settings are:

- LEFT MARGIN 5 spaces
- TOP MARGIN 3 lines
- BOTTOM MARGIN 3 lines
- PAGE LENGTH 66 lines (11 inches)
- Display the report on the screen

If you include an OUTPUT section, you must place it immediately before the SELECT section in your report specification.

The FORMAT Section

The FORMAT section includes instructions for printing a custom report. Page headers, column headings, rows of data, and page trailers can be included in this section if a custom report is desired, or it can contain instructions for a simple, default report.

FORMAT for a Default Report

If you do not want a custom report, you can use the EVERY ROW statement in the FORMAT section in place of the control blocks. If you use an EVERY ROW statement, you cannot use any control blocks or other statements in the FORMAT section. INFORMIX-SQL will use a default format for your report, displaying every column and row that the SELECT statement selected. Following is the FORMAT section of an INFORMIX-SQL report specification that uses an EVERY ROW statement:

```
format                {format for a default report}
  every row
end
```

FORMAT for a Custom Report

The **FORMAT** section of a custom report is divided into seven control blocks. You can combine one or more of the control blocks in a variety of ways to tailor your reports. The order of these control blocks in the **FORMAT** section of a report specification is not significant. Seven control blocks are available:

- **FIRST PAGE HEADER**
- **PAGE HEADER**
- **PAGE TRAILER**
- **ON EVERY ROW**
- **ON LAST ROW**
- **BEFORE GROUP OF**
- **AFTER GROUP OF**

Creating Page Headers and Trailers

You can print titles and column headings at the top of each page of a report. You can also print footer information at the bottom of each page. Use the following control blocks in your **FORMAT** section to control these report features:

- **FIRST PAGE HEADER**
- **PAGE HEADER**
- **PAGE TRAILER**

Where Headers and Trailers Appear

Text you specify with the **PAGE HEADER** control block appears on the line immediately under the top margin on every page of a report. Text you specify with the **PAGE TRAILER** control block appears at the bottom of each page, on the line immediately above the bottom margin.

If you want different text to appear on the first page of a report, use the **FIRST PAGE HEADER** control block to specify headings for the first page only.

Printing a Report Heading

Use the PAGE HEADER control block to specify the text you want in the heading on every page. Enter the text exactly as you want it to appear, and enclose it in quotation marks. You can also specify where you want the header positioned.

For example, you may want to print a report title at the top of the page like this:

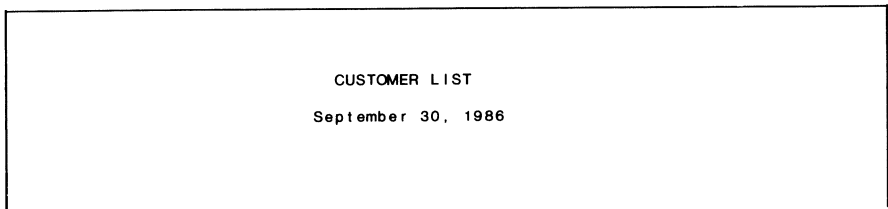


Figure 8-7 shows the PAGE HEADER control block that produces this heading.

```
page header
  print column 20, "CUSTOMER LIST"
  skip 1 line
  print column 18, "September 30, 1986"
```

Figure 8-7. A Sample PAGE HEADER Control Block

Following is a description of what each of the keywords in the sample PAGE HEADER control block does.

PRINT Use the PRINT statement to specify the text you want to print and the spacing you want to use.

Enclose the text you want to print in quotation marks.

If you want to control spacing, enter the number of spaces you want and the keyword SPACES.

For example, to indent text 20 spaces, enter **print 20 spaces**. You can also position text with the **COLUMN** keyword.

COLUMN Use the **COLUMN** keyword to place text in specific column positions. You can also position text with the **SPACES** keyword.

Note: The **COLUMN** keyword operates with respect to the Left Margin. If the Left Margin is 5 spaces and you enter

print column 15

printing will actually begin in column 20.

SKIP Use the **SKIP** statement to specify how many blank lines you want to leave. If you want to advance to the top of the next page, use **SKIP TO TOP OF PAGE**.

Printing Column Headings

You can also use the **PAGE HEADER** or **FIRST PAGE HEADER** control blocks to specify headings you want to appear above the columns of a report. For example, you might want the following report title and column headings to appear in the report:

CUSTOMER LIST		
September 30, 1986		
Last Name	Company	City

To print the title and the column headings, use a **PAGE HEADER** control block like the one shown in Figure 8-8.

```
page header
  print column 20, "CUSTOMER LIST"
  skip 1 line
  print column 18, "September 30, 1986"
  skip 2 lines
  print "Last Name",      {do not forget}
  column 18, "Company",    {the commas}
  column 40, "City"
  skip 2 lines
```

Figure 8-8. Column Headings in a PAGE HEADER Control Block

Note that commas must be used to separate column keywords and the data that are to appear on the same line.

Numbering Pages Automatically

You can include a page number in either the header or trailer by using a PAGENO expression in a PRINT statement. When you print the report, each page is numbered automatically.

The following PAGE TRAILER control block produces text and a page number at the bottom of each page:

```
page trailer
  print "Confidential Information",
  column 55, pageno using "Page ##"
```

(Additional information about the keyword USING appears in the section “Controlling the Appearance of Numbers,” later in this chapter.)

This control block produces the following page trailer:

Confidential Information

Page 1

Printing Rows of Data

To print the rows of information in the body of a report, you must use the ON EVERY ROW control block in your FORMAT section. The ON EVERY ROW control block lists the names of the columns that contain the data you want in the report. (Do not confuse the ON EVERY ROW control block with the EVERY ROW statement.)

You can only print data from the columns selected in the SELECT section. For example, if you do not include the **fname** column in the SELECT section, you cannot list it in the ON EVERY ROW control block of the FORMAT section.

Figure 8-9 shows a report specification that uses an ON EVERY ROW control block. The ON EVERY ROW control block lists the same column names as the SELECT statement.

```
database
  stores
end

output
  left margin 10
end

select
  lname, company, city
  from customer
  order by city
end

format

page header
  print column 20, "CUSTOMER LIST"
  skip 1 line
  print column 18, "September 30, 1986"
  skip 2 lines
  print "Last Name",
  column 18, "Company",
  column 40, "City"
  skip 2 lines

on every row
  print lname,
  column 18, company,
  column 40, city

page trailer
  print "Confidential Information",
  column 55, pageno using "Page ##"

end
```

Figure 8-9. Using the ON EVERY ROW Control Block

The specification in Figure 8-9 produces a report that prints the last name, first name, and company for every customer.

Here is what a portion of the report looks like:

CUSTOMER LIST		
September 30, 1986		
Last Name	Company	City
Vector	Los Altos Sports	Los Altos
Lawson	Runners & Others	Los Altos
Beatty	Sportstown	Menlo Park
Grant	Gold Medal Sports	Menlo Park

Confidential Information	Page 1
--------------------------	--------

Organizing the Data

List the column names in the order that you want the information printed in the ON EVERY ROW control block. For example, if you want to alter the above report to print the company, followed by last name and city, list the column names in an ON EVERY ROW control block as follows:

```
format
  on every row
  print company,
    column 18, lname,
    column 40, city
end
```

Grouping Data

You can divide the information in a report into groups. For example, you might want to group customers by city, as illustrated in Figure 8-10.

CUSTOMER LIST		
September 30, 1986		
Last Name	Company	City
Vector Lawson	Los Altos Sports	Los Altos
	Runners & Others	Los Altos
Beatty Grant	Sportstown	Menlo Park
	Gold Medal Sports	Menlo Park
Watson Parmelee	Watson & Son	Mountain View
	Olympic City	Mountain View
Baxter	Blue Ribbon Sports	Oakland
Currie Ream	Phil's Sports	Palo Alto
	Athletic Supplies	Palo Alto
Higgins	Play Ball!	Redwood City
Quinn	Quinn's Sports	Redwood City
Jaeger	AA Athletics	Redwood City
Albertson	Sporting Place	Redwood City
Sipes	Kids Korner	Redwood City
Sadler	Sports Spot	San Francisco
Pauli Miller Keyes	All Sports Supplies	Sunnyvale
	Sport Stuff	Sunnyvale
	Sports Center	Sunnyvale

Confidential Information

Page 01

Figure 8-10. A Report with Data Grouped by City

The Group Control Blocks

To group the information in a report, use the following control blocks in your **FORMAT** section:

- **BEFORE GROUP OF**
- **AFTER GROUP OF**

The **BEFORE GROUP OF** and **AFTER GROUP OF** control blocks specify what **INFORMIX-SQL** does when it gets to the beginning or end of a group. Enter the column name that contains the information you want to group by. The same column name must also appear in the **ORDER BY** clause in the **SELECT** section.

The instructions you include in a **BEFORE GROUP OF** control block apply to actions **INFORMIX-SQL** performs *before* printing the next group of rows. You can use a **BEFORE GROUP OF** control block to print headings above a group of rows.

The instructions you include in the **AFTER GROUP OF** control block apply to actions **INFORMIX-SQL** performs *after* printing a group of rows. You can use an **AFTER GROUP OF** control block to calculate totals or subtotals under a group of rows.

For example, if you want to group the information in a report by city, first order by city in the **SELECT** section. Then group by city in a **BEFORE GROUP OF** or **AFTER GROUP OF** control block in the **FORMAT** section.

Figure 8-11 shows a report specification that groups information by city. As shown in the figure, you must use the same column (**city**) to order *and* group the information.

```
database
  stores
end

output
  left margin 10
end

select
  lname, company, city
  from customer
  order by city      {order by city}
end

format

page header
  print column 20, "CUSTOMER LIST"
  skip 1 line
  print column 18, "September 30, 1986"
  skip 2 lines
  print "Last Name",
  column 18, "Company",
  column 40, "City"
  skip 2 lines

on every row
  print lname,
  column 18, company,
  column 40, city

after group of city  {group by city}
  skip 2 lines

page trailer
  print "Confidential Information",
  column 55, pageno using "Page ##"

end
```

Figure 8-11. Grouping Information with an AFTER GROUP OF Clause

This specification produces the “Customer List” report shown in Figure 8-10 (at the beginning of the section “Grouping Data”). The report groups together all the customers located in the same city.

Calculations in a Report

Often you want to perform calculations on the numeric data you include in a report. For example, you may want to total the values in a column, multiply the values in a column by a constant, or add the values in one column to the values in another.

INFORMIX-SQL provides arithmetic operators, built-in math functions (also called aggregates), and group functions you can include in a report specification to compute values in reports.

Arithmetic Operators

You can perform addition, subtraction, multiplication, division, or exponentiation on any of the numeric columns in a report. Use any of the following arithmetic operators in the FORMAT section:

Symbol	Function
+	Addition
—	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Example of Arithmetic Operations

If you want to calculate a 15 percent increase in the price for every item in the **stock** table, you could enter the following ON EVERY ROW control block in the FORMAT section:

```
on every row
  print stock_num,
    column 15, manu_code,
    column 30, description,
    column 50, unit,
    column 57, unit_price * 1.15
  using "$$$$$. $$"
```

When you run the report, **INFORMIX-SQL** multiplies the current unit price (**unit_price**) by 1.15 and prints the results in the report.

Here is what a portion of the report would look like:

1	HRO	baseball gloves	case	\$287.50
1	HSK	baseball gloves	case	\$920.00
1	SMT	baseball gloves	case	\$517.50
2	HRO	baseball	case	\$144.90
3	HSK	baseball bat	case	\$276.00
4	HSK	football	case	\$1104.00

Using Parentheses

You can use parentheses in calculations to control the order in which **INFORMIX-SQL** computes results. For example, if you want to divide the values in a column by 15 percent of 75, then add 100 to the result, enter

```
print unit_price / (.15 * 75) + 100
using "$$$$$.$$"
```

Controlling the Appearance of Numbers

To format numbers, include a **PRINT USING** statement in the **FORMAT** section. The **PRINT USING** statement shows the format you want to use for the numbers. You can format numbers to include dollar signs and commas, like this

```
$5,970
$8,520
$5,550
```

by using the statement

```
PRINT col1 USING "$,$$$"
```

Note: If the columns in a table are the **MONEY** data type, **INFORMIX-SQL** automatically formats the numbers with dollar signs.

The symbols between the quotation marks in the following **PRINT USING** statement are the pattern that **INFORMIX-SQL** uses to format numbers in the **unit_price** column.

```
print unit_price * 1.15 using "####"
```

Using Fill Characters

The example above uses pound symbols (####) as *fill characters*. A collection of fill characters between quotation marks is a *format string*. You use a format string to set up a number format. When **INFORMIX-SQL** prints a report, it substitutes actual numbers in place of the format string.

In the example, the format string tells **INFORMIX-SQL** that every number should include four places. Thus, if the numbers in the report are 45, 456, and 4560, **INFORMIX-SQL** prints them as

```
    45
   456
  4560
```

INFORMIX-SQL prints leading blanks so that every number contains four places. When you use the # fill character, **INFORMIX-SQL** adds as many leading blanks as necessary so that every number is right-justified.

If a number is larger than the pattern you set up in the **PRINT USING** statement, **INFORMIX-SQL** prints a group of asterisks instead of the number. If the numbers are 45, 456, 4560, and 45600, they appear in the report as

```
    45
   456
  4560
 *****
```

The Fill Characters

There are several special fill characters you can use with format strings:

Character	Function
#	Prints leading blanks if a number does not completely fill the format (as shown in the preceding examples).

- &** Prints leading zeros if a number does not completely fill the format (for example, 0045 or 0456).
- Prints a leading minus sign if the number is negative. If you use just one minus sign, and the number is negative, **INFORMIX-SQL** will place a minus sign in the position you indicate in the format string.

If you use more than one minus sign, **INFORMIX-SQL** places it immediately to the left of the most significant digit. (This is often called a “floating” minus sign.)
- *** Prints leading asterisks (****45** or ***456**) if a number does not completely fill the format (useful for printing checks).
- <** Prints numbers left-justified.

You can combine these symbols in a variety of ways to control and print numbers in the format you want. Refer to Chapter 5 of the *INFORMIX-SQL Reference Manual* for a complete explanation of all the number formatting features available in **INFORMIX-SQL**.

Adding a Dollar Sign

If you want to include a dollar sign and a comma with the numbers, include those symbols in the **PRINT USING** statement, like this:

```
print unit_price * 1.15 using "$#,###"
```

Now **INFORMIX-SQL** prints the numbers like this:

```
$    45
$   456
$4,560
```

Making the Dollar Sign Float

In the preceding example, the dollar sign always appears in the same position because the # fill character tells **INFORMIX-SQL** to print blanks if there are no numbers. It fixes the dollar sign in a specific position in the number.

If you want the dollar sign to float, or change position according to the length of the number, enter the **PRINT USING** statement like this:

```
print unit_price * 1.15 using "$$, $$$"
```

Now the dollar sign appears at the beginning of the numbers, regardless of their length, like this:

```
    $45  
   $456  
 $4,560
```

Math Functions

You can perform calculations using any of the following six built-in math functions (or aggregates):

TOTAL OF	Adds the values in a column for all rows selected.
MIN OF	Prints the smallest (MINimum) value in a column for all rows selected.
MAX OF	Prints the largest (MAXimum) value in a column for all rows selected.
AVERAGE OF	Calculates the average of the values in a column for all rows selected.
PERCENT	Calculates the percentage of all rows selected that are in a group (refer to Chapter 5 of the <i>INFORMIX-SQL Reference Manual</i> for details).
COUNT	Counts the number of rows in the table or group you specify.

Totaling a Column

The following is a sample report that lists information about selected items in the **stock** table. In addition, it indicates the total number of these items on order. (The report specification that produced this report is shown in Figure 8-12.)

Stock Number	Manufacturer	Description	Unit	Quantity
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	10
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	1
5	NRG	tennis racquet	each	10
5	NRG	tennis racquet	each	5
5	SMT	tennis racquet	each	5
Total Quantity on Order:				46

```
database
  stores
end

select
  items.stock_num, items.manu_code,
  quantity, description, unit
  from items, stock
  where items.stock_num
     = stock.stock_num and
     items.manu_code
     = stock.manu_code
     and items.stock_num = 5
end
format

page header
  print "Stock Number",
  column 15, "Manufacturer",
  column 30, "Description",
  column 50, "Unit",
  column 60, "Quantity"
  skip 2 lines

on every row
  print stock_num,
  column 15, manu_code,
  column 30, description,
  column 50, unit,
  column 60, quantity

on last row
  skip 1 line
  print column 20, "Total Quantity on Order: ",
  column 55, total of quantity using "###"

end
```

Figure 8-12. Calculating the Total of a Column

To add all the orders, use the ON LAST ROW control block with a PRINT statement that contains the TOTAL OF function. Enter TOTAL OF, followed by the name of the column you want to total.

Counting Rows in a Report

You can use COUNT to tally the number of rows that INFORMIX-SQL has selected. For example, the next report prints the number of orders, as well as the total quantity of each order, where the stock number is 5. (The report specification that produced this report is shown in Figure 8-13.)

Stock Number	Manufacturer	Description	Unit	Quantity
5	NRG	tennis racquet	each	10
5	NRG	tennis racquet	each	5
5	SMT	tennis racquet	each	5
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	10
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	5
5	ANZ	tennis racquet	each	1
Total Quantity on Order:				46
Total Number of Orders:				8

```

database
    stores
end

select
    items.stock_num, items.manu_code,
    quantity, description, unit
from items, stock
where items.stock_num
= stock.stock_num and
items.manu_code
= stock.manu_code
and items.stock_num = 5
end
format

page header
    print "Stock Number",
    column 15, "Manufacturer",
    column 30, "Description",
    column 50, "Unit",
    column 60, "Quantity"
    skip 2 lines

on every row
    print stock_num,
    column 15, manu_code,
    column 30, description,
    column 50, unit,
    column 60, quantity

on last row
    skip 1 line
    print column 20, "Total Quantity on Order: ",
    column 55, total of quantity using "###"
    skip 1 line
    print column 20, "Total Number of Orders: ",
    column 55, count using "###"
end

```

Figure 8-13. Counting the Number of Rows

The ON LAST ROW control block prints the number of orders along with the total quantity of each order.

Combining Arithmetic Operators and Math Functions

You can use both arithmetic operators and math functions in the statements of the **FORMAT** section. You may want to find the average of a column and then multiply the average by a constant. To do this, combine **AVERAGE** with the multiplication (*) arithmetic operator.

For example, to average the shipping charges (contained in the **ship_charge** column in the **orders** table) and then multiply the result by 1.15, enter the following in your **FORMAT** section:

```
on last row
  skip 1 line
  print (average of ship_charge) * 1.15
```

Group Functions

In addition to arithmetic and math functions, **INFORMIX-SQL** contains functions you can use to perform calculations on groups of information in a report. The group functions allow you to calculate, for example, subtotals or interim results.

The group functions you can use in your report specifications include

- **GROUP TOTAL OF**
- **GROUP COUNT**
- **GROUP MIN OF**
- **GROUP MAX OF**
- **GROUP PERCENT**
- **GROUP AVERAGE OF**

Any time you use a group function for a column, you must **ORDER BY** that column in the **SELECT** section.

To perform calculations on groups of rows in your report, use the group functions in the **AFTER GROUP OF** control block of the **FORMAT** section.

For example, the report statement in Figure 8-14 sorts items by stock number and prints the total quantity on order for each item.

In the example, the **stock__num** column in the **items** table is given the display label **stockno**. ACE orders and groups the output by the values in the **stockno** column.

The use of a display label in this manner is described in the “**SELECT Section**” section in Chapter 5 of the *INFORMIX-SQL Reference Manual*.

```

database
    stores
end

output
    left margin 0
end

select
    items.stock_num stockno, items.manu_code,
    description, unit, quantity
    from items, stock
    where items.stock_num =
    stock.stock_num and
    items.manu_code =
    stock.manu_code
order by stockno
end

format

page header
    print "Stock Number",
    column 15, "Manufacturer",
    column 30, "Description",
    column 50, "Unit",
    column 60, "Quantity"
    skip 2 lines

on every row
    print stockno,
    column 15, manu_code,
    column 30, description,
    column 50, unit,
    column 60, quantity

after group of stockno
    skip 1 line
    print column 20, "Total Quantity on Order: ",
    column 55, group total of quantity using "###"
    skip 2 lines

end

```

Figure 8-14. AFTER GROUP OF and GROUP TOTAL OF Functions in a Report

Here is a portion of the report generated by the specification in Figure 8-14:

Stock Number	Manufacturer	Description	Unit	Quantity
1	HRO	baseball gloves	case	1
1	HRO	baseball gloves	case	1
1	HRO	baseball gloves	case	1
1	SMT	baseball gloves	case	1
1	SMT	baseball gloves	case	1
1	HSK	baseball gloves	case	1
Total Quantity on Order:				6
2	HRO	baseball	case	1
2	HRO	baseball	case	1
Total Quantity on Order:				2
3	HSK	baseball bat	case	1
3	HSK	baseball bat	case	1
3	HSK	baseball bat	case	1
Total Quantity on Order:				3
4	HSK	football	case	1
4	HRO	football	case	1
4	HSK	football	case	1
4	HRO	football	case	1
Total Quantity on Order:				4

Displaying a Report

Report output can be displayed in several ways:

- Displayed on your screen
- Printed on a printer
- Written to a file
- Piped as input to another program (on UNIX systems only)

You specify how you want the report routed in the OUTPUT section of your report specification. You can also use the OUTPUT section to change the margins and page length.

The OUTPUT section in a report specification must appear immediately before the SELECT section.

Printing the Report on a Printer

To print the report on a printer, use the REPORT TO PRINTER statement in the OUTPUT section.

Figure 8-15 shows an OUTPUT section that prints the report on a printer. In addition, it changes the left margin to 10 and the page length to 50.

```
database
  company
end

output
  left margin 10
  page length 50
  report to printer
end

select lname, company, city
  from customer
end

format
  every row
end
```

Figure 8-15. An OUTPUT Section that Prints a Report

Writing the Report to a File

To send the report results to a file, use the **REPORT TO *filename*** statement in the OUTPUT section, enclosing the name of the file in quotation marks.

If you want to write the report to a file named **cust__rep**, you would enter the OUTPUT section as

```
output
  report to "cust__rep"
end
```

When you run the report, **INFORMIX-SQL** creates a file named **cust__rep** and writes the report results in it.

Piping the Report to Another Program

Refer to Chapter 5 of the *INFORMIX-SQL Reference Manual* for a complete explanation of piping report results as input to another program. (This option is available only on UNIX systems.)

Chapter Summary

- A report writer allows you to select and format information from a database. The **INFORMIX-SQL** report writer is called **ACE**.
- The first step of generating a report is creating a report specification. A report specification tells **INFORMIX-SQL** what database the report is based on (the **DATABASE** section), what information is to be selected from what table or tables (the **SELECT** section), and how to format the report (the **FORMAT** section).
- To create a report specification, first select **Report** from the **INFORMIX-SQL** Main Menu and then select **Generate** from the **REPORT** Menu.

If you have not selected a current database, **INFORMIX-SQL** will ask you to select one after you select **Generate**.

- Choose a name for your report and enter it when the **GENERATE REPORT** Screen appears.
- Select a table from the list displayed by the next menu. **INFORMIX-SQL** will use this table to create a default report specification.

You can use the default specification as is, modify it, or write your own specification using the **New** option from the **REPORT** Menu.

INFORMIX-SQL automatically compiles the default report specification.

- If you want to change the default specification, select **Modify** from the **REPORT** Menu. Select the editor you want to use from the **USE-EDITOR** Screen. Once you select an editor, you will see the default specification on your screen.

- Change the specification as you desire and then exit from the editor.
- The next step in generating a report is compiling the specification. After you exit from the editor, **INFORMIX-SQL** will display the **MODIFY REPORT** Menu. Select **Compile** to compile your report specification.
- If your specification contains errors, **INFORMIX-SQL** will display the **COMPILE REPORT** Menu. Select **Correct** to edit and correct your specification. After correcting the specification, exit from the editor and compile the specification again.
- Once you have successfully compiled the specification, select **Save-and-exit**. **INFORMIX-SQL** will display the **REPORT** Menu. The report is now available for you to use with **INFORMIX-SQL**.
- The final step in generating a report is to run the compiled report specification. Select **Run** from the **REPORT** Menu. Choose the name of the report you want to run from the **RUN REPORT** Screen.

Chapter 9

Database Structure and Integrity

Chapter 9 Table of Contents

Chapter Overview	5
Changing the Structure of a Database	6
Removing a Table	6
Dropping a Table with RDSQL.....	6
Dropping a Table from the TABLE Menu.....	7
Removing an Index.....	7
Renaming a Table.....	7
Removing a Database	8
Dropping a Database with RDSQL.....	8
Dropping a Database from the DATABASE Menu.....	9
Changing the Structure of a Table.....	9
A Look at the customer Table.....	10
Using RDSQL.....	10
Renaming a Column	11
The ALTER TABLE Statement.....	11
Dropping a Column from a Table.....	13
Changing the Data Type of a Column.....	14
Combining Keywords in the Statement	16
Using the Schema Editor.....	16
Adding a Column to a Table.....	16
Modifying a Column in a Table	17
Dropping a Column.....	18
Transactions.....	19
Audit Trails	20
Comparing Audit Trails and the Transaction Log	20
Views	21
Data Types	22
CHAR Columns.....	22
Numeric Columns	23
DECIMAL Columns	24
SMALLINT and INTEGER Columns	24
SMALLFLOAT and FLOAT Columns	25

SERIAL Columns	25
DATE Columns.....	26
MONEY Columns.....	26
Column and Row Lengths	27
Creating Indexes	29
When to Index Columns	29
Indexing Guidelines	30
Columns You Can Index	30
Single-Column Indexes.....	31
Multiple-Column Indexes.....	31
Columns You Should Not Index	32
Preventing Duplicate Entries	32
Ascending and Descending Indexes	33
Clustered Indexes.....	33
Chapter Summary.....	34

Chapter Overview

This chapter explains how to modify the structure of an **INFORMIX-SQL** database and table. It also describes ways to ensure database integrity, and discusses data types and indexes. The following topics are considered:

- Deleting a database
- Removing a table or index
- Renaming a table
- Renaming, adding, or dropping a column from a table
- Protecting the integrity of your database with transactions
- Protecting the integrity of a table with audit trails
- Creating dynamic “windows” on your database with views
- Selecting the appropriate data type for a column
- Calculating column and row length in a table
- Using strategies when indexing columns

Changing the Structure of a Database

You can change the structure of a database by

- Removing a table
- Removing an index
- Renaming a table
- Removing the database

You must have the appropriate database or table privileges to perform these operations. See the “Data Access Statements” and the “Grant Statement” sections in Chapter 2 of the *INFORMIX-SQL Reference Manual* for more information.

Before you rename a table or remove a table or index, be sure the database containing the table or index is the current database.

Removing a Table

You remove a table by running an **RDSQL DROP TABLE** statement or with the **Drop** Option on the TABLE Menu.

Caution! Be careful when you drop a table. **INFORMIX-SQL** deletes the table you specify, *along with all the data it contains*. If you accidentally delete the wrong table, you must recreate the table and restore all the data from a backup copy; if you do not have a backup, you must reenter the data.

Dropping a Table with RDSQL

Use the DROP TABLE statement to remove a table from a database. The general format is

DROP TABLE *table-name*

where *table-name* is the name of the table you want to remove.

Dropping a Table from the TABLE Menu

The **Drop** option on the TABLE Menu removes a table from the database. When you select the **Drop** option, **INFORMIX-SQL** prompts you to enter the name of the table you want to drop. You are then asked to confirm that you do want to drop the table. If you change your mind, you can press N and the table is untouched. Press Y and **INFORMIX-SQL** drops the table from the database.

Removing an Index

Use the **RDSQL DROP INDEX** statement to remove an index. The general format is

```
DROP INDEX index-name
```

where *index-name* is the name of the index you want to remove.

If you accidentally remove the wrong index, you can easily recreate it with the **CREATE INDEX** statement.

Renaming a Table

Use the **RENAME TABLE** statement to change the name of an existing table. The general format is

```
RENAME TABLE oldname TO newname
```

where *oldname* is the current name of the table and *newname* is the name you want to assign to it.

Note: When you rename a table you have referred to in reports, screen forms, or **RDSQL** command files, you need to change those references after you rename the table. For reports and screen forms, you should update the specifications and then recompile them. For **RDSQL** command files, you can just change the references.

Removing a Database

You remove a database by running an **RDSQL DROP DATABASE** statement or with the **Drop** Option on the DATABASE Menu.

You must own all the tables in the database or have DBA status to drop a database.

If you want to drop the current database, you must first run the **RDSQL CLOSE DATABASE** statement to close it. The format of this statement is

```
CLOSE DATABASE
```

Caution! Be very careful when you drop a database. **INFORMIX-SQL** deletes all tables, indexes, and data from the database. **INFORMIX-SQL** also removes the database directory if all the files in the directory belong to the database. If the database directory contains files that do not belong to the database, the **INFORMIX-SQL** removes only the database files and leaves the rest of the directory intact.

Dropping a Database with RDSQL

Use the **DROP DATABASE** statement to remove an entire database. The general format is

```
DROP DATABASE database-name
```

where *database-name* is the name of the database you want to delete.

Dropping a Database from the DATABASE Menu

The **Drop** option on the DATABASE Menu removes a database. When you select the **Drop** option, **INFORMIX-SQL** prompts you to enter the name of the database you want to drop. You are then asked to confirm that you do want to drop the database. If you change your mind, press **N** and the database is untouched. Press **Y** and **INFORMIX-SQL** drops the database.

Changing the Structure of a Table

You can change the structure of a table at any time, even if you have stored data in the table. You can

- Rename a column
- Add a column
- Delete a column
- Change the data type of a column

You can make any of these changes to a table that you create. You cannot change the structure of a table created by another person unless you are granted **ALTER** privilege for that table.

You can only change the structure of tables that belong to the current database. If you try to change a table that does not belong to the current database, **RDSQL** displays an error message.

You can change the structure of a table by using an **RDSQL** statement or by using the table schema editor available from the **TABLE** Menu. The first part of this section describes the use of **RDSQL** statements. The second part demonstrates the use of the table schema editor.

A Look at the customer Table

The examples in this section are based on the **customer** table from the demonstration database. Here is the CREATE TABLE statement that created the **customer** table:

```
create table customer
(
    customer__num    serial(101),
    fname            char(15),
    lname            char(15),
    company           char(20),
    address1          char(20),
    address2          char(20),
    city              char(15),
    state             char(2),
    zipcode           char(5),
    phone             char(18)
);
```

Using RDSQL

The following sections show you how to rename a column with the RENAME COLUMN statement and how to add a column, drop a column, or change the data type of a column with the ALTER TABLE statement.

After running each **RDSQL** statement, you can use the **Info** option on the RDSQL Menu to inspect the change made to the structure of the table.

Renaming a Column

Use the RENAME COLUMN statement to change the name of a column. The general format is

```
RENAME COLUMN table.oldcol TO newcol
```

For example, to rename the **customer__num** column in the **customer** table to **cust__number**, enter

```
rename column customer.customer__num  
to cust__number
```

Note: When you rename a column, indexes and privileges are modified automatically to include the new name. However, you will have to change references to the column in all form specifications, report specifications, or **RDSQL** command files. Once you update the form and report specifications, you must recompile them.

The ALTER TABLE Statement

You can use the ALTER TABLE statement to add a column to a table, delete a column from a table, or change the data type of a column.

Adding a Column. Use the ALTER TABLE statement with the ADD keyword to add one or more new columns to a table at any time, even when the table already contains data.

Adding a New Column at the End of a Table. The following statement adds a column for credit rating at the end of the **customer** table:

```
ALTER TABLE customer  
ADD (credit_rate char(3))
```

When you run this statement, **RDSQL** appends the **credit__rate** column to the list of columns.

You can add as many columns to the end of the table as you want, as long as each column name is unique. If you want to add two columns for general information to the end of the **customer** table, for example, you could enter this statement:

```
ALTER TABLE customer
  ADD (gen_info1 char(40),
       gen_info2 char(40))
```

Default Values. When you add columns to a table, **RDSQL** assigns a NULL value to each row for each new column, by default.

For information on how to avoid NULLs, see Appendix I (**dbupdate**) in the *INFORMIX-SQL Reference Manual*.

Inserting a New Column in a Table. You can use a **BEFORE** clause with the **ADD** keyword of the **ALTER TABLE** statement to insert a new column before an existing column. **RDSQL** automatically adjusts indexes and authorizations after a column is inserted so that the rest of the table remains intact.

The following statement inserts the **county** column before the **state** column of the **customer** table:

```
ALTER TABLE customer
  ADD (county char(25) BEFORE state)
```

You can insert as many columns in the table as you want. The following statement adds the **manager** column before the **address1** column and the **manage_phone** column before the **phone** column:

```
ALTER TABLE customer
  ADD (manager char(20) BEFORE address1,
       manage_phone char(18) BEFORE phone)
```

You may have to add references to the new column in your form specifications, report specifications, or **RDSQL** command files. Once you update the form and report specifications, you must recompile them.

A table cannot contain two columns with the same name. If you accidentally add a column in the wrong location, you will have to delete it first (see the section “Dropping a Column from a Table,” which follows) and then add the column again in the correct location.

Dropping a Column from a Table

Use the **ALTER TABLE** statement with the **DROP** keyword to drop a column and any data it contains. **RDSQL** automatically adjusts indexes and authorizations after a column is dropped so that the rest of the table remains intact.

The following statement drops the **manage__phone** column (added previously) from the **customer** table:

```
ALTER TABLE customer
  DROP (manage__phone)
```

As with other **ALTER TABLE** keywords, you can work with more than one column at a time. The following example drops the **gen__info1** and **gen__info2** columns (added previously) from the **customer** table:

```
ALTER TABLE customer
  DROP (gen__info1, gen__info2)
```

Use the **DROP** keyword when you are sure you have no need for the data in a column, or when you have just added a new column in the wrong location.

Note: When you drop a column, you will have to drop references to the column in all form specifications, report specifications, or **RDSQL** command files. Once you update the form and report specifications, you must recompile them.

Changing the Data Type of a Column

You can use the **MODIFY** keyword of the **ALTER TABLE** statement to change the data type of a column.

You normally change a column's data type when you need to store larger values than what is possible with the current data type. For example, if you create a **SMALLINT** column and find later on that you need to store integers larger than 32,767, you will need to change the data type for the column to **INTEGER**. The following statement changes the data type of the **item_num** column in the **items** table from **SMALLINT** to **INTEGER**:

```
ALTER TABLE items
    MODIFY (item_num INTEGER)
```

Similarly, you can use the **MODIFY** keyword of the **ALTER TABLE** statement to increase the length of a **CHAR** column. The following statement, for example, changes the length of the **company** and **city** columns in the **customer** table to 30 characters and 20 characters, respectively:

```
ALTER TABLE customer
    MODIFY (company char(30),
           city char(20))
```

When you run an **ALTER TABLE** statement with the **MODIFY** keyword, **RDSQL** converts values from the old data type to the new one.

Converting Numeric to Numeric. When you convert columns from one numeric data type to another, you may occasionally find round-off errors. For example, if you convert a **FLOAT** column to **DECIMAL(4,2)**, **RDSQL** rounds off the floating-point numbers before storing them as decimal numbers.

Caution! Be very careful with conversions in which the new data type cannot store values as large as the old data type. If **RDSQL** encounters a value that does not fit the new data type, it displays an error message and leaves the table untouched by the **ALTER TABLE** statement.

For example, you might want to convert a column from the **INTEGER** to the **SMALLINT** data type. Assume the following values exist in the column:

100 400 700 50000 700

RDSQL stops when it encounters the number 50,000 and leaves the table untouched by the **ALTER TABLE** statement.

The same situation may occur if you attempt to transfer data from **FLOAT** or **SMALLFLOAT** columns to **INTEGER**, **SMALLINT**, or **DECIMAL** columns.

Converting Numeric to CHAR. When you convert the values in a column from numeric to **CHAR**, you can no longer use them as easily for calculations or sorting.

You can convert a **CHAR** column to a numeric column, but if the **CHAR** column contains any characters that are not valid in a numeric column (such as the letter *l* instead of the number *1*), **RDSQL** cannot complete the **ALTER TABLE** statement. If the statement does not run successfully, **RDSQL** displays an error message and leaves the column values as characters.

Combining Keywords in the Statement

You can use two or more of the keywords (ADD, DROP, MODIFY) in a single statement. The keywords can appear in any order, separated by commas, and will run sequentially. For example, the following statement changes the data type of the **county** column and drops the **manager** column (added earlier):

```
ALTER TABLE customer
    MODIFY (county char(30)),
    DROP (manager)
```

If RDSQL cannot run part of an ALTER TABLE statement, it ignores the entire statement and leaves the table unchanged.

Using the Schema Editor

The **Alter** option on the TABLE Menu can be used to make changes to the structure of a table.

Adding a Column to a Table

The **Alter** option on the TABLE Menu is used to add a column to a table schema. The process of adding a column is identical to that used to create a table with the schema editor.

1. Select the **Alter** option on the TABLE Menu.
INFORMIX-SQL asks for the name of the table you want to change. Select the table, and the ALTER TABLE Menu is displayed. Use the **ARROW** keys to move the highlight to the part of the schema where you want to position the new column. The new column appears immediately above the highlight.
2. Select the **Add** option. Enter the information necessary to add the column. When you finish, press **RETURN**.

INFORMIX-SQL again displays the ALTER TABLE Menu.

3. Select the **Exit** option. Then select **Build-new-table** to make the table addition permanent. INFORMIX-SQL makes the changes to the database table and returns you to the TABLE Menu.

Modifying a Column in a Table

The **Alter** option on the TABLE Menu is also used to change a column definition in the table schema. You can change one part of a column or several parts.

1. Select the **Alter** option on the TABLE Menu. INFORMIX-SQL asks for the name of the table you want to change. Select the table, and the ALTER TABLE Menu is displayed. Use the **ARROW** keys to move the highlight to the part of the schema you want to change.
2. Select the **Modify** option. INFORMIX-SQL displays the appropriate screen for the part of the schema you highlighted. Make the desired changes. If you want to modify another part of the schema, use the **ARROW** keys to move the highlight. Make menu selections by using the **SPACEBAR** to highlight the option you want and by pressing **RETURN**. The screen changes as you move the highlight.

If your changes decrease the length of a column, the **MODIFY ANYWAY** Menu appears, with a warning that you may lose data. Select **Yes** if you want to continue with the change or **No** if you do not.

When you finish your changes, press **RETURN**. INFORMIX-SQL displays the ALTER TABLE Menu.

3. Select the **Exit** option. Then select **Build-new-table** to make the change permanent. INFORMIX-SQL makes the changes to the database table and returns you to the TABLE Menu.

Dropping a Column

You can remove any column from the schema. Use the **Drop** option on the ALTER TABLE Menu when you decide a column is no longer useful.

1. Select the **Alter** option from the TABLE Menu.
INFORMIX-SQL asks for the name of the table you wish to change. Select the table, and the ALTER TABLE Menu is displayed. Use the **ARROW** keys to move the highlight to the column you want to delete. Select the **Drop** option by pressing the **d** key.

INFORMIX-SQL displays the REMOVE Menu, with a message reminding you that this change will delete data when you select **Build-new-table**.

```
REMOVE clients : ☐ Yes ☒ No
Deletes the highlighted column from the table.

----- Page 1 of 1 ----- mydata ----- Press CONTROL-W for Help -----

Column Name          Type          Length  Index Nulls
-----
customer num         Serial              Unique No
fname                Char              15      Yes
lname                Char              15      Yes
company              Char              20      Yes
address1              Char              20      Yes
address2              Char              20      Yes
city                  Char              15      Yes
state                 Char               2      Yes
zipcode              Char               5      Dups  Yes
phone                 Char              18      Yes

-17072: Any data in this column will be lost when you select Exit, Build.
```

2. Select the **Yes** option from the REMOVE Menu if you want to delete the column; select **No** and the table is untouched. If you select the **Yes** option, INFORMIX-SQL removes the column from the screen and returns to the ALTER TABLE Menu.
3. Select **Exit**. Then select **Build-new-table**. If you decide you do not want to remove the column, select the **Discard-new-table** option. INFORMIX-SQL restores the table schema to its prior state.

Transactions

A transaction is a series of operations on a database that you want **INFORMIX-SQL** to complete entirely or not at all. For example, if you are writing an accounting program, you may often want **INFORMIX-SQL** to perform several operations as a unit to ensure a correct balance. Multiple tables may be affected in the course of completing a series of updates or inserts, and data in the tables must remain “in balance.” Transactions give you the ability to treat multiple operations as a single transaction. By using transactions, you can ensure the integrity of the data throughout the database.

Transactions have the following characteristics:

- To use transactions with a database, you must create a transaction log for the database. A transaction log is a file in which **INFORMIX-SQL** records all modifications to the database.
- With transactions you can execute a group of statements as a unit.
- If you are satisfied that the group of statements has produced the desired result, you can commit the transaction to the database. If you are not happy with the results, you can discard the transaction without modifying the database.
- With transactions you can also treat individual statements as a single transaction. This means that **INFORMIX-SQL** commits each statement if it is successfully executed. If the statement fails to run to completion, **INFORMIX-SQL** discards the transaction and does not modify the database.

For complete information about transactions and the **RDSQL** statements used with transactions, refer to Chapter 2, “The **RDSQL** Query Language,” in the *INFORMIX-SQL Reference Manual*.

Audit Trails

An audit trail is a file that contains a history of all additions, deletions, updates, and manipulations to a database table. An audit trail serves a purpose similar to that of a transaction log: each is used to maintain a record of modifications to a database, and each can be used to update backup copies of a database.

Comparing Audit Trails and the Transaction Log

Audit trails compare with transactions and a transaction log in the following respects:

- **INFORMIX-SQL** records all modifications to a *database* in a transaction log. An audit trail records modifications to a single *table*.
- To use transactions, you must create a database “with transactions;” the transaction log cannot be easily added and dropped as desired. Audit trails may be added and deleted as needed.
- With audit trails, you can record modifications to a single important table without incurring the system expense of maintaining a transaction log on the entire database.
- Unlike transactions, audit trails do not protect against statements that fail to run to completion.

For complete information about the use of audit trails, refer to Chapter 2, “The RDSQL Query Language,” in the *INFORMIX-SQL Reference Manual*.

Views

In certain instances, you may want a user to work with only part of a database, especially if it contains information that you want to remain confidential. With **INFORMIX-SQL**, you can control what the user sees by creating a dynamic window on the database called a *view*.

Views have the following two advantages:

- Since a view resembles a table in many respects, you can substitute a view name for a table name in most **INFORMIX-SQL** statements. This means that you can allow the user to insert, select, update, and delete information through a view.
- You can create a view so that **INFORMIX-SQL** makes sure that any data entered or updated by the user satisfies one or more preconditions.

For example, you can create a view that allows the user to work only with customers in California. The user may not query for or enter data about a customer who lives in Oregon. This way, you can prevent a user from entering information that will not be accessible through the view.

For complete information about the uses of views, refer to Chapter 2, “The RDSQL Query Language,” in the *INFORMIX-SQL Reference Manual*.

Data Types

Each column in a table is assigned a data type. The data type selected should match the kind of information the column will store.

There are four categories of data types:

Character Character columns store any combination of letters, numbers, and symbols.

Numeric There are six different numeric data types. Each is designed to store a different kind of numeric data.

Date The date data type stores calendar dates.

Money Money columns store currency amounts.

These data types are described in detail in the following sections.

CHAR Columns

CHAR (short for CHARacter) columns store any combination of letters, numbers, and symbols. You normally use CHAR columns to store names, addresses, social security numbers, job titles, project names, and other information that combines letters, numbers, and symbols.

You must specify the length of each CHAR column. This length is the total number of characters you want to reserve for the name, address, job title, or whatever data you wish to store in the column.

The maximum length of a CHAR column is 32767 bytes.

Caution! You can store numbers in CHAR columns, but if you do, you cannot use them in some arithmetic operations. If you want to add a series of numbers together, for example, you should store the numbers with one of the numeric data types. Numbers that will not be used in arithmetic operations, such as zip codes, phone numbers, social security numbers, and so on, should be placed in CHAR columns. Numbers that have leading zeros (such as some zip codes) will have the zeros stripped if they are stored as type INTEGER and so should also be placed in CHAR columns.

Numeric Columns

There are six data types for numbers. Each is designed for a different kind of number. You cannot store characters or symbols in any numeric column, although you can use plus signs (+) and minus signs (–) to show whether numbers are positive or negative.

Figure 9-1 summarizes the numeric data types.

Data Type	Kind of Number
DECIMAL	Numbers with definable scale and precision
SMALLINT	Whole numbers from –32,767 to +32,767
INTEGER	Whole numbers from –2,147,483,647 to +2,147,483,647
SMALLFLOAT	Single-precision, floating-point numbers with approximately seven significant digits
FLOAT	Double-precision, floating-point numbers with approximately 14 significant digits
SERIAL	Sequential integers assigned by INFORMIX-SQL

Figure 9-1. Numeric data types

Small Machine Features: On some small machines, SMALLFLOAT and FLOAT numbers are treated as DECIMAL numbers with 8 or 16 significant digits.

DECIMAL Columns

DECIMAL columns store numbers with the number of digits you specify. When you create a DECIMAL column, you specify the total number of significant digits, up to a maximum of 32.

The range of numbers you can store is from 10×10^{-128} to 10×10^{126} , although only 32 digits are significant. You may also specify the number of digits after the decimal point. The general format is

decimal [(*precision* [,*scale*])]

where *precision* is the total number of digits you want, and *scale* is the number of digits to the right of the decimal point. Specifying the precision and scale is optional. If you do not specify precision, DECIMAL is treated as DECIMAL(16), a floating decimal with a precision of 16 places. If you do not specify scale, INFORMIX-SQL treats the column as having a floating-point decimal.

For example, the phrase (6,2) indicates that the column stores six-digit numbers, with four digits before the decimal point and two after the decimal point.

SMALLINT and INTEGER Columns

SMALLINT and INTEGER columns store whole numbers—numbers that have no fractional portion. SMALLINT columns store whole numbers from $-32,767$ to $+32,767$. INTEGER columns store whole numbers from $-2,147,483,647$ to $+2,147,483,647$.

SMALLFLOAT and FLOAT Columns

SMALLFLOAT and FLOAT columns store binary floating-point numbers. SMALLFLOAT columns contain single-precision, floating-point numbers with approximately 7 significant digits. FLOAT columns contain double-precision, floating-point numbers with approximately 14 significant digits. FLOAT columns do not necessarily store larger numbers (this depends on the computer you are using); they store numbers with greater precision.

The number you enter in a floating-point column and the number **INFORMIX-SQL** displays may differ slightly, depending on how your computer internally stores floating-point numbers.

On some small machines, SMALLFLOAT and FLOAT are treated as DECIMAL(8) and DECIMAL(16).

SERIAL Columns

Serial numbers ensure that a unique number is assigned to each row of the table. You do not enter data in a SERIAL column. Each time you add a new row to a table, **INFORMIX-SQL** automatically assigns the next number, in sequence, to the SERIAL column. The default starting number is one, but you can select any number greater than zero as your starting number. The highest serial number **INFORMIX-SQL** can assign is 2,147,483,647.

Once **INFORMIX-SQL** assigns a SERIAL number, you cannot change it. (If you were allowed to change the number, **INFORMIX-SQL** could not guarantee uniqueness.) You can, however, use the INSERT statement to insert a value into a serial column, as long as it does not duplicate any existing values. Each table in a database can contain only one SERIAL column.

DATE Columns

Use the DATE data type for columns in which you want to store calendar dates. A DATE column holds a date in the form *mm/dd/yyyy* where *mm* is the month (1–12), *dd* is the day of the month (1–31), and *yyyy* is the year (0001–9999).

DATE value	Meaning
05/02/1985	May 2, 1985
09/08/1883	September 8, 1883
12/25/1900	December 25, 1900

MONEY Columns

Money columns store currency amounts. When you create a MONEY column, you specify the total number of significant digits up to a maximum of 32.

The general format for a MONEY column is

```
money [( precision [,scale] )]
```

where *precision* is the total number of digits you want, and *scale* is the number of digits to the right of the decimal point. MONEY columns are always treated as fixed decimal numbers. If no precision or scale is given, a MONEY column defaults to DECIMAL(16,2).

Column and Row Lengths

Although there is no logical limit to the length of a row in an **INFORMIX-SQL** table, there may be a system-dependent limit. You may find it useful to know how to compute the length in *bytes* of a column and a row. A byte is equivalent to a single character. You control the length of a **CHAR** column by the length you assign to it in the **CREATE TABLE** statement. **INFORMIX-SQL** assigns lengths to other types of columns as follows:

Data Type	Length
CHAR	length specified with CREATE TABLE
SMALLINT	2 bytes
INTEGER	4 bytes
SMALLFLOAT	4 bytes
FLOAT	8 bytes
SERIAL	4 bytes
DATE	4 bytes
MONEY	varies (see below)
DECIMAL	varies (see below)

Small Machine Features: On some small machines, **SMALLFLOAT** and **FLOAT** are implemented as **DECIMAL(8)** and **DECIMAL(16)**, respectively.

The number of bytes required for a **DECIMAL** or **MONEY** column is half the precision you specify in the **CREATE TABLE** statement plus 1. If you specify the following data type for a column,

decimal (14,4)

the total storage required is eight bytes ($1 + 14/2$).

You can easily calculate the length of each row in a table. To do so, you add the lengths for each column.

Following is the CREATE TABLE statement that was used to create the **orders** table in the demonstration database.

```
create table orders
(order_num      serial(1001),
 order_date     date,
 customer_num   integer,
 ship_instruct  char(40),
 backlog       char(1),
 po_num        char(10),
 ship_date     date,
 ship_weight    decimal(8,2),
 ship_charge    money(6),
 paid_date     date
)
```

To calculate the length of a row in the **orders** table:

Qty	Type	Length Each	Length Total
1	SERIAL	4 bytes	4 bytes
3	DATE	4 bytes	12 bytes
1	INTEGER	4 bytes	4 bytes
1	CHAR	40 bytes	40 bytes
1	CHAR	1 byte	1 byte
1	CHAR	10 bytes	10 bytes
1	DECIMAL	5 bytes (1 + 8/2)	5 bytes
1	MONEY	4 bytes (1 + 6/2)	4 bytes

total length of each row: 80 bytes

Each row in the **orders** table uses 80 bytes.

Once you enter and run a CREATE TABLE command, you can ask **INFORMIX-SQL** to calculate the length of the rows automatically. Use the **Info** option on the RDSQL Menu or the TABLE Menu. **INFORMIX-SQL** asks which table you want information about. After you supply the table name, select the **Status** option. **INFORMIX-SQL** lists data about the table, including the length of the rows.

Creating Indexes

Creating indexes for the columns in a table will, under certain circumstances, help **INFORMIX-SQL** find information more quickly. However, just as you can find information in a book without consulting the index, **INFORMIX-SQL** can find information in a database even if you do not index any columns.

In fact, if there is only one table in your database, you probably will not need any indexes until you have entered several hundred rows of data into the table. Too many indexes can slow down your system. As your database increases in size and your needs change, you can always add and drop indexes.

When to Index Columns

INFORMIX-SQL does not limit the number of indexes you create; if you wanted to, you could create an index for every column. It usually makes sense to index only certain columns.

Indexes considerably speed up some **INFORMIX-SQL** operations, especially searching for data in a large database and sorting it. For example, if you generally sort personnel information alphabetically by last name, it makes sense to index the column that holds last names.

On the other hand, when you create indexes, it takes slightly longer to enter and modify data. Each time you enter a new row or modify an existing one, **INFORMIX-SQL** needs to update both the table and the index. Furthermore, indexes take up computer disk space.

You should decide which columns to index by considering these factors in light of the particular requirements of your database. The following guidelines should make this decision easier.

Indexing Guidelines

If your database contains only one table, do not index any columns until the file contains at least 200 rows.

Index Join Columns. If your database contains more than one table, you should have at least one join column in each table. You should index any column used in a join operation.

When an index is not placed on the columns included in a join field, **PERFORM** issues a warning. The warning tells the user that queries can be performed more quickly with the index.

Auto-Indexing. If you execute a **SELECT** statement that includes a join between two tables and there are no indexes on the joined columns, **RDSQL** creates a temporary index on the table with the larger number of rows before performing the join. The index disappears when the query finishes. Auto-indexing is transparent to the user and results in a dramatic improvement in the speed of unindexed queries.

Index Search and Sort Columns. Other indexing choices depend on how you use your database. After indexing join columns, you may want to use the database for a while, until you know what searches and sorts you perform regularly. Then you can add indexes for these columns.

Columns You Can Index

You can create an index that applies to a single column or to several columns as a group.

Single-Column Indexes

Single-column indexes speed database queries and sorts that involve the indexed column.

For example, if you frequently sort a table of names by last name, you should index the last-name column. With the index, it would take less time for **INFORMIX-SQL** to sort the names like this:

Last Name	First Name
Smith	Dan
Smith	Laura
Smith	Amy
Smith	Erin
Sullivan	Winston
Sullivan	Meredith
Sullivan	Ann
Sullivan	Joe

Multiple-Column Indexes

You can also create an index that applies to more than one column. Such multiple-column indexes help **INFORMIX-SQL** sort several columns together.

If you wanted to sort the above list so that all the first names that go with a single last name are in alphabetical order, you would create an index that applies to both the last-name and first-name columns. Then **INFORMIX-SQL** could quickly sort the names like this:

Last Name	First Name
Smith	Amy
Smith	Dan
Smith	Erin
Smith	Laura
Sullivan	Ann
Sullivan	Joe
Sullivan	Meredith
Sullivan	Winston

You could sort these names whether or not you create an index. Creating an index simply speeds up the process if you are working with a large database.

Columns You Should Not Index

You cannot create an index for a column or group of columns whose total length exceeds 120 bytes. The section “Column and Row Lengths,” earlier in this chapter, includes more information about column length.

You generally want to avoid indexing columns that contain a large number of duplicate values. In this situation, an index can significantly slow the process of deleting data and changing the values in the indexed column.

For example, it is seldom useful to index a column that contains either a Y (for yes) or an N (for no). For very large databases, indexing zip code and state columns is also not appropriate, as each column can contain thousands of identical entries. An index on a column with more than 60,000 identical records slows down a search and could cause an **INFORMIX-SQL** error.

Preventing Duplicate Entries

INFORMIX-SQL normally allows you to enter the same value in different rows of an indexed column. In other words, even if you index a **last_name** column, you can still enter information about many people whose last name is Smith.

You may sometimes want to prevent users from entering duplicate information in an indexed column. If you create a column to store social security numbers, you might want to prevent duplicate entries. Use the **RDSQL** create unique index statement to prevent duplicate entries in a column.

Ascending and Descending Indexes

Unless you instruct otherwise, **RDSQL** creates indexes in ascending order—that is, from *A* to *Z* for **CHAR** fields, from low to high for numeric and **MONEY** fields, and from earlier to later in time for **DATE** fields. You can also create descending indexes.

You should create descending indexes only for fields you intend to regularly sort in descending order. For example, you might create a descending index for a **DATE** field if you normally sort the data in a table in reverse chronological order.

For example, to create a descending index on the **paid_date** column in the **orders** table, use this **RDSQL** statement:

```
create index i_pd_date on orders (paid_date desc)
```

Clustered Indexes

Since both **UNIX** and **DOS** extract information from the disk in blocks, the more rows that are physically on the same block and that are already in the same order as an index, the faster an indexed retrieval proceeds. Ordinarily, no relationship need exist between the physical order of the data in the **dat** file and the order in an index. You can, at least temporarily, cause the physical order in the table to be the same as the order in an index through *clustering*. Chapter 2 of the *INFORMIX-SQL Reference Manual* discusses clustered indexes in more detail.

Chapter Summary

- **INFORMIX-SQL** provides both **RDSQL** statements and Menu options for dropping databases, tables, and indexes, altering tables, and renaming tables and columns.
- Use transactions if you want **INFORMIX-SQL** to complete a series of operations entirely or not at all.
- Audit trails are available to ensure the integrity of a database table.
- You can create a dynamic window called a view that restricts the user's access to information in a database.
- You can calculate the length of each row manually, or let **INFORMIX-SQL** do it for you once you create the table.
- The data type assigned to a column should match the information that the column will store.
- Under certain circumstances, indexes help **INFORMIX-SQL** find information more quickly.
- A dynamic approach to using indexes gives you the best results.

List of Appendixes

Appendix A. The stores Database

Appendix B. Environment Variables

Appendix C. INFORMIX-SQL Reserved Words

Appendix A

The *stores* Database

The *stores* Database

This appendix contains four sections.

- The first section describes the structure of each table in the **stores** database, presents the **RDSQL** statement used to create each table, and notes any indexes on columns in each table.
- The second section presents a map of the **stores** database with potential join columns identified.
- Section three discusses the join columns that link the five tables in the database and presents figures illustrating these relationships.
- The final section shows the data contained in each table in the **stores** database.

Structure of the Tables

The **stores** database contains information about a fictitious sporting goods distributor that services stores in the Western United States. The database is made up of the following five tables:

- customer
- orders
- items
- stock
- manufact

customer

The **customer** table contains information about 18 stores that order sporting goods from the distributor. This information includes the name of the store representative and the store name, address, and telephone number. The columns of the **customer** table are as follows:

customer__num	serial(101)
fname	char(15)
lname	char(15)
company	char(20)
address1	char(20)
address2	char(20)
city	char(15)
state	char(2)
zipcode	char(5)
phone	char(18)

The **customer__num** column is indexed as unique. The **zipcode** column is indexed to allow duplicate values.

orders

The **orders** table contains information about orders placed by the distributor's customers. This information includes the order number, date the order was made, the customer number, shipping instructions, whether a backlog exists, the customer's

purchase order number, date the order was shipped, weight of the order, shipment charge, and the date the customer paid for the order. The columns of **orders** are as follows:

order__num	serial(1001)
order__date	date
customer__num	integer
ship__instruct	char(40)
backlog	char(1)
po__num	char(10)
ship__date	date
ship__weight	decimal(8,2)
ship__charge	money(6)
paid__date	date

The **order__num** column is indexed and must contain unique values; the **customer__num** column is indexed but allows duplicates.

items

The **items** table keeps track of individual items in an order. For example, some orders only contain one item, while other orders contain as many as five items. Information in the **items** table includes the item number, order number, stock number, manufacturing code, quantity, and the total price for each item that has been ordered. The columns of the **items** table are as follows:

item__num	smallint
order__num	integer
stock__num	smallint
manu__code	char(3)
quantity	smallint
total__price	money(8)

The **order__num** column is indexed and allows duplicate values. A multiple-column index for the **stock__num** and **manu__code** columns also permits duplicate values.

stock

The distributor offers customers 15 different types of sporting goods. For example, the distributor offers baseball gloves from three different manufacturers and basketballs from one manufacturer.

The **stock** table is a catalog of the items sold by the distributor. For each item in stock, the **stock** table lists a stock number identifying the type of item, a code identifying the manufacturer, a description of the item, its unit price, the unit by which the item must be ordered, and a description of the unit (for example, a case containing 10 baseballs).

The **stock** table contains the following columns:

stock__num	smallint
manu__code	char(3)
description	char(15)
unit__price	money(6)
unit	char(4)
unit__descr	char(15)

A multiple-column index for the **stock__num** and **manu__code** columns allows only unique values.

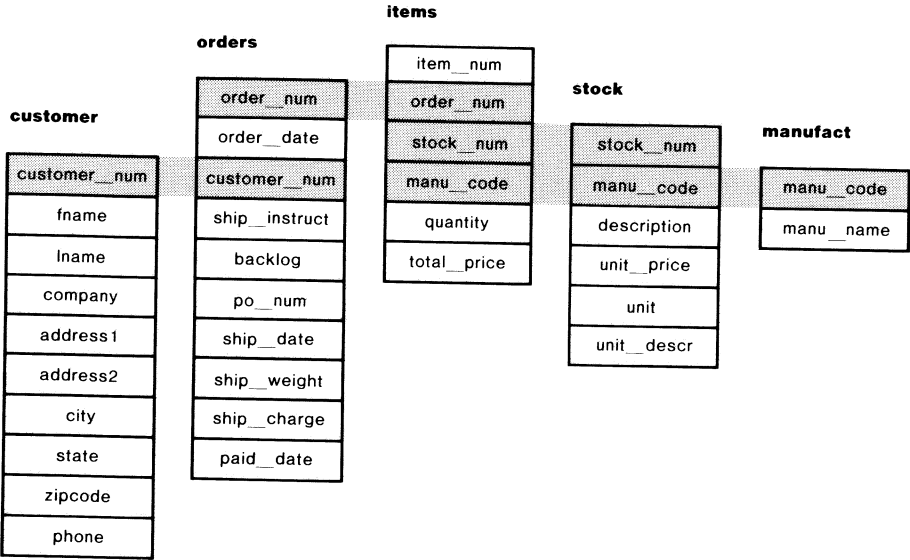
manufact

The distributor handles sporting goods from five manufacturers. Information about these manufacturers is kept in the **manufact** table. This information consists of an identification code and the manufacturer's name. The columns in the **manufact** table are as follows:

manu__code	char(3)
manu__name	char(15)

The **manu__code** column is indexed and must contain unique values.

Map of the stores Database



Join Columns That Link the Database

The five tables of the **stores** database are linked together by join columns. This section identifies these columns and describes how you can use them to retrieve data from several tables at once and display it as if it were stored in a single table. Figures show the relationships among the tables and how information stored in one table supplements information in other tables.

Join Columns in the *customer* and *orders* Tables

The **customer** table and the **orders** table are joined by the **customer__num** column, as shown in the following example:

customer table (detail)		
customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins

orders table (detail)		
order_num	order_date	customer_num
1001	01/20/1986	104
1002	06/01/1986	101
1003	10/12/1986	104
1004	04/12/1986	106

Figure A-1. Tables Joined by the **customer_num** Column

For example, the **customer** table contains a **customer_num** column that holds a number identifying the customer, along with columns for name, company, address, and telephone number. The row with information about Anthony Higgins contains the number 104 in the **customer_num** column. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. According to Figure A-1, customer 104 (Anthony Higgins) has placed two orders, since his customer number appears in two rows of the **orders** table.

The join relationship lets you select information from both tables. This means you can retrieve Anthony Higgins's name and address and information about his orders at the same time.

Join Columns in the *orders* and *items* Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If a particular order includes several items, the same order number appears in several rows of the **items** table. Figure A-2 shows this relationship.

			orders table (detail)	
order_num	order_date	customer_num		
1001	01/20/1986	104		
1002	06/01/1986	101		
1003	10/12/1988	104		
			items table (detail)	
item_num	order_num	stock_num	manu_code	
1	1001	1	HRO	
1	1002	4	HSK	
2	1002	3	HSK	
1	1003	9	ANZ	
2	1003	8	ANZ	
3	1003	5	ANZ	

Figure A-2. Tables Joined by the **order_num** Column

Join Columns in the *items* and *stock* Tables

The **items** table and the **stock** table are joined by two columns: the **stock__num** column stores a stock number for an item, and the **manu__code** column stores a code that identifies the manufacturer. You need both the stock number and the manufacturing code to uniquely identify an item. For example, the item with the stock number 1 and the manufacturing code HRO is a Hero baseball glove, while the item with the stock number 1 and the manufacturing code HSK is a Husky baseball glove.

The same stock number and manufacturing code may appear in more than one row of the **items** table if the same item belongs to separate orders, as shown in Figure A-3.

items table (detail)			
item__num	order__num	stock__num	manu__code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock table (detail)		
stock__num	manu__code	description
1	HRO	baseball glove
1	HSK	baseball glove
1	SMT	baseball glove

Figure A-3. Tables Joined by Two Columns

Join Columns in the *stock* and *manufact* Tables

The **stock** table and the **manufact** table are joined by the **manu__code** column. The same manufacturing code may appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. This relationship is illustrated in Figure A-4.

stock__num	manu__code	description
1	HRO	baseball glove
1	HSK	baseball glove
1	SMT	baseball glove
2	HRO	baseball

stock table
(detail)

manu__code	manu__name
NRG	Norge
HSK	Husky
HRO	Hero

manufact table
(detail)

Figure A-4. Tables Joined by the **manu__code** Column

Joining lets you rearrange your view of a database whenever you want. It provides flexibility that lets you create new relationships between tables without redesigning the database. You can easily expand the scope of a database by adding new tables that join the existing tables. As you read through this manual, you will see programs that set up the join relationships across tables of the **stores** database. You can refer to the preceding figures whenever you need to review these relationships.

*Data in the **stores** Database*

The data in the **stores** database is displayed in the tables that follow.

customer Table

customer_num	name	name	company	address1	address2	city	state	zipcode	phone
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075
102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543
104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	415-356-9876
108	Donald	Quinn	Quinn's Sports	587 Alvarado		Redwood City	CA	94063	415-544-8729
109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789
110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	415-743-3611
111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022	415-887-7235
113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	415-356-9982
114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	415-886-6677
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	415-356-1123
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	415-534-8822
117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	415-245-4578
118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	415-655-0011

orders Table

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1001	01/20/86	104	ups	n	B77836	02/01/86	20.40	10.00	03/22/86
1002	06/01/86	101	po on box, deliver back door only	n	9270	06/06/86	50.60	15.30	07/03/86
1003	10/12/86	104	via ups	n	B77890	10/13/86	35.60	10.80	11/04/86
1004	04/12/86	106	ring bell twice	y	8006	04/30/86	95.80	19.20	
1005	12/04/86	116	call before delivering	n	2865	12/19/86	80.80	16.20	12/30/86
1006	09/19/86	112	after 10 am	y	Q13557		70.80	14.20	
1007	03/25/86	117		n	278693	04/23/86	125.90	25.20	
1008	11/17/86	110	closed Monday	y	LZ230	12/06/86	45.60	13.80	12/21/86
1009	02/14/86	111	door next to supersaver	n	4745	03/04/86	20.40	10.00	04/21/86
1010	05/29/86	115	deliver 776 Gary if no answer	n	429Q	06/08/86	40.60	12.30	07/22/86
1011	03/23/86	104	ups	n	B77897	04/13/86	10.40	5.00	06/01/86
1012	06/05/86	117		n	278701	06/09/86	70.80	14.20	
1013	09/01/86	104	via ups	n	B77930	09/18/86	60.80	12.20	10/10/86
1014	05/01/86	106	ring bell, kick door loudly	n	8052	05/10/86	40.60	12.30	07/18/86
1015	07/10/86	110	closed Mon	n	MA003	08/01/86	20.60	6.30	08/31/86

items Table

item__num	order__num	stock__num	manu__code	quantity	total__price
1	1001	1	HRO	1	250.0
1	1002	4	HSK	1	960.0
2	1002	3	HSK	1	240.0
1	1003	9	ANZ	1	20.0
2	1003	8	ANZ	1	840.0
3	1003	5	ANZ	5	99.0
1	1004	1	HRO	1	960.0
2	1004	2	HRO	1	126.0
3	1004	3	HSK	1	240.0
4	1004	1	HSK	1	800.0
1	1005	5	NRG	10	280.0
2	1005	5	ANZ	10	198.0
3	1006	6	SMT	1	36.0
4	1005	6	ANZ	1	48.0
1	1006	5	SMT	5	125.0
2	1006	5	NRG	5	190.0
3	1006	5	ANZ	5	99.0
4	1006	6	SMT	1	36.0
5	1006	6	ANZ	1	48.0
1	1007	1	HRO	1	250.0
2	1007	2	HRO	1	126.0
3	1007	3	HSK	1	240.0
4	1007	4	HRO	1	480.0
5	1007	7	HRO	1	600.0
1	1008	8	ANZ	1	840.0
2	1008	9	ANZ	5	100.0
1	1009	1	SMT	1	450.0
1	1010	6	SMT	1	36.0
2	1010	6	ANZ	1	48.0
1	1011	5	ANZ	5	99.0
1	1012	8	ANZ	1	840.0
2	1012	9	ANZ	10	200.0
1	1013	5	ANZ	1	19.8
2	1013	6	SMT	1	36.0
3	1013	6	ANZ	1	48.0
4	1013	9	ANZ	2	40.0
1	1014	4	HSK	1	960.0
2	1014	4	HRO	1	480.0
1	1015	1	SMT	1	450.0

stock table

stock__num	manu__code	description	unit__price	unit	unit__descr
1	HRO	baseball gloves	250.00	case	10 gloves / case
1	HSK	baseball gloves	800.00	case	10 gloves / case
1	SMT	baseball gloves	450.00	case	10 gloves / case
2	HRO	baseball	126.00	case	24 / case
3	HSK	baseball bat	240.00	case	12 / case
4	HSK	football	960.00	case	24 / case
4	HRO	football	480.00	case	24 / case
5	NRG	tennis racquet	28.00	each	each
5	SMT	tennis racquet	25.00	each	each
5	ANZ	tennis racquet	19.80	each	each
6	SMT	tennis ball	36.00	case	24 cans / case
6	ANZ	tennis ball	48.00	case	24 cans / case
7	HRO	basketball	600.00	case	24 / case
8	ANZ	volleyball	840.00	case	24 / case
9	ANZ	volleyball net	20.00	each	each

manufact Table

manu__code	manu__name
ANZ	Anza
HSK	Husky
HRO	Hero
NRG	Norge
SMT	Smith

Appendix B

Environment Variables

Environment Variables

INFORMIX-SQL makes the following assumptions about the user's environment:

- The field separator for unloaded data files is the vertical bar (| = ASCII 124).
- The editor used is the predominant editor for the operating system (usually **vi** for UNIX systems and **edlin** for DOS systems).
- The database worked with is in the current directory.
- If the computer is running UNIX, the program that sends files to the printer is **lp**. If the computer is running DOS, the name of the printer device is **lpt1**.
- On UNIX systems, you use **/tmp** to store temporary files. On DOS systems, you use the current directory to store temporary files.
- The **INFORMIX-SQL** program and its associated files are located in **/usr/informix** (on a UNIX system) or **\informix** (on a DOS system).

Setting Environment Variables

You can change any of the assumptions by setting one or more of the environment variables recognized by **INFORMIX-SQL**. These environment variables are listed by operating system in later sections of this appendix.

Setting Environment Variables on UNIX Systems

You may set environment variables at the system prompt or in your **.profile** (Bourne shell) or **.login** (C shell) file. If you set an environment variable at the system prompt, you will have to assign it again the next time you log onto the system. If you set a variable in your **.profile** (Bourne shell) or **.login** (C shell) file, UNIX will assign it automatically every time you log onto the system.

If you are using the Bourne shell, enter the following two commands to set the **ABCD** environment variable to *value*:

```
ABCD=value
export ABCD
```

If you are using the C shell, enter the following command to set the **ABCD** environment variable to *value*:

```
setenv ABCD value
```

Setting Environment Variables on DOS Systems

You may set environment variables at the system prompt or in your **AUTOEXEC.BAT** file. If you set an environment variable at the system prompt, you will have to assign it again the next time you start the system. If you set a variable in your **AUTOEXEC.BAT** file, DOS will assign it automatically every time you start the system.

Enter the following command to set the **ABCD** environment variable to *value*:

```
set ABCD=value
```

UNIX Environment Variables

The environment variables recognized by **INFORMIX-SQL** are

DBDELIMITER Specifies the field delimiter used in unloaded data files for use with the **LOAD** and **UNLOAD** statements in **RDSQL**. The vertical bar (| = ASCII 124) is the default.

For example, if you are using the C shell, enter the following command to set the field delimiter to a plus sign:

```
setenv DBDELIMITER +
```

DBDATE Specifies the format you want to use for **DATE** values. Through **DBDATE**, you can specify

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year

The default value for **DBDATE** is

```
MDY4/
```

where **M** represents the month, **D** represents the day, **Y4** represents a four-digit year, and the separator is a (/) slash (mm/dd/yyyy).

Other acceptable values for the separator are a hyphen (-) or a period (.). The slash appears if you attempt to use any values other than the hyphen or period.

Always specify the separator value last. Always follow the Y with the number of digits specified for the year. To make the date appear as mm.dd.yy, set the DBDATE environment variable for the C shell as follows:

```
setenv DBDATE MDY2.
```

For the Bourne shell, use

```
DBDATE=MDY2.  
export DBDATE
```

Suppose you wanted the date to appear in European format (dd-mm-yyyy). For the C shell, you would set the DBDATE environment variable as follows:

```
setenv DBDATE DMY4—
```

For the Bourne shell, you would use

```
DBDATE=DMY4—  
export DBDATE
```

DBEDIT

Names the text editor you want to use to modify form specifications files, report specification files, and command files from within INFORMIX-SQL. For most systems, the default is **vi**.

DBMENU

Specifies the user-menu that **INFORMIX-SQL** accesses. The default is to access the Main Menu of the user-menu for the current database. If you want to make **reports** your default menu, enter the following command for the C shell:

```
setenv DBMENU reports
```

For the Bourne shell, use

```
DBMENU=reports  
export DBMENU
```

DBMONEY

Applies to **MONEY** values. It has the format

[front][. | ,][back]

where *front* is the optional symbol that precedes the **MONEY** value, the comma or the period is the optional symbol that separates the integral from the fractional part of the **MONEY** value, and *back* is the optional symbol that follows the **MONEY** value. The *front* and *back* symbols may be up to seven characters long and may contain any characters except commas or periods.

The default value for **DBMONEY** is

\$.

where the dollar sign precedes the **MONEY** value, and the period (.) separates the integral from the fractional part of the **MONEY** value. (No *back* symbol appears.)

Suppose you wanted to represent MONEY values in Deutsche Marks. For the C shell, set the DBMONEY environment variable as follows:

```
setenv DBMONEY DM,
```

For the Bourne shell, use

```
DBMONEY=DM,  
export DBMONEY
```

where DM is the currency symbol, and the comma separates the integral from the fractional part of the MONEY value.

When you make a change to the *back* symbol, you must also supply the *front* symbol and the value separator (comma or period). Similarly, if you change the value separator from a comma to a period, you must supply the *front* symbol.

DBPATH

Specifies a list of directories for **INFORMIX-SQL** to search (in addition to the current directory) for databases and associated files. If you have not selected a database, **INFORMIX-SQL** looks to DBPATH to determine the list of available databases. Once you have selected a database, **INFORMIX-SQL** looks first in the current directory and then in the parent directory of the directory containing the database for the associated forms, reports, and command files (recall that each database is contained in a separate directory).

For example, if you want **INFORMIX-SQL** to search for database files in Kevin's and Debbie's directories, as well as in your current directory, you would specify

```
DBPATH=/usr/Kevin:/usr/Debbie
export DBPATH
```

(Make sure you enter a colon between the directory names.)

- | | |
|-------------|--|
| DBPRINT | Specifies the print program for your computer. For most UNIX systems, the default program is lp . |
| DBTEMP | Specifies the directory into which INFORMIX-SQL should place its temporary files. You need not set DBTEMP if /tmp is satisfactory. |
| INFORMIXDIR | Specifies the directory containing the INFORMIX-SQL files. |

DOS Environment Variables

The DOS environment variables recognized by **INFORMIX-SQL** are as follows:

DBDELIMITER Specifies the field delimiter used in unloaded data files for use with the **LOAD** and **UNLOAD** statements in **RDSQL**. The vertical bar (| = ASCII 124) is the default.

To change the field delimiter to a semicolon, for example, you can enter

```
set DBDELIMITER=;
```

DBDATE Specifies the format you want to use for **DATE** values. Through **DBDATE**, you can specify

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year

The default value for **DBDATE** is

MDY4/

where **M** represents the month, **D** represents the day, **Y4** represents a four-digit year, and the separator is a (/) slash (mm/dd/yyyy). Other acceptable values for the separator are a hyphen (-) or a period (.). The slash appears if you attempt to use any values other than the hyphen or period.

Always specify the separator value last. Always follow the Y with the number of digits specified for the year. To make the date appear as mm.dd.yy, set the DBDATE environment variable as follows:

```
set DBDATE=MDY2.
```

Suppose you wanted the date to appear in European format (dd-mm-yyyy). You would set the DBDATE environment variable as follows:

```
set DBDATE=DMY4—
```

DBEDIT

Names the text editor you want to use to modify form specifications files, report specification files, and command files from within INFORMIX-SQL. The default editor is **edlin**. To change the default editor, you can enter

```
set DBEDIT=program-name
```

where *program-name* is the name of the new editor.

DBMENU

Specifies the user-menu that INFORMIX-SQL accesses. The default is to access the Main Menu of the user-menu for the current database. If you want to make **reports** your default menu, enter the following:

```
set DBMENU=reports
```

DBMONEY

Applies to MONEY values. It has the format

```
[front]{. | ,}[back]
```

where *front* is the optional symbol that precedes the MONEY value, the comma or the period is the optional symbol that separates the integral from the fractional part of the

MONEY value, and *back* is the optional symbol that follows the MONEY value. The *front* and *back* symbols may be up to seven characters long and may contain any characters except commas or periods.

The default value for DBMONEY is

\$.

where the dollar sign precedes the MONEY value, and the period separates the integral from the fractional part of the MONEY value. (No *back* symbol appears.)

If you want to represent MONEY values in Deutsche Marks, for example, you would set the DBMONEY variable as follows

```
set DBMONEY=DM,
```

where DM is the currency symbol, and the comma separates the integral from the fractional part of the MONEY value.

When you make a change to the *back* symbol, you must also supply the *front* symbol and the value separator (comma or period). Similarly, if you change the value separator from a comma to a period, you must supply the *front* symbol.

DBPATH

Specifies a list of directories for **INFORMIX-SQL** to search for databases and associated files. If you have not selected a database, **INFORMIX-SQL** looks to DBPATH to determine the list of available databases. If you have selected a database, **INFORMIX-SQL** looks first in the current directory and then in the directory where your database is located for associated forms, reports, and command files.

For example, if you want **INFORMIX-SQL** to search for database files in Kevin's directory on C: and Debby's directory on D:, you can enter

```
set DBPATH=c:\kevin;d:\debby
```

(When you set the DBPATH variable, make sure you enter a semicolon between directory names.)

Make sure you set the DBPATH variable before you execute the **startsql** command.

DBPRINT Specifies the name of the printer device. The default is **lpt1**. To change the default to **lpt2**, for example, you can enter

```
set DBPRINT=lpt2
```

DBTEMP Specifies the directory where **INFORMIX-SQL** should place its temporary files.

INFORMIXDIR Specifies the directory containing the **INFORMIX-SQL** files.

Appendix C

INFORMIX-SQL Reserved Words

INFORMIX-SQL Reserved Words

The following words are **INFORMIX-SQL** reserved words and cannot be used as program identifiers.

access	count	format
add	create	from
after	database	grant
all	date	group
allowing	day	having
alter	dba	header
and	decimal	headings
any	declare	help
as	default	if
asc	define	in
attribute	delete	include
attributes	delimiters	index
autonext	desc	info
average	display	input
avg	displayonly	insert
before	distinct	instructions
begin	do	integer
bell	downshift	into
between	drop	is
bottom	editadd	joining
by	editupdate	last
char	else	left
check	end	length
clipped	every	let
close	exclusive	like
column	exists	line
columns	exitnow	lineno
comment	fetch	lines
comments	first	load
commit	float	lock
composites	for	log
connect	foreach	lookup
continue	form	margin

master	public	synonym
matches	query	systables
max	queryclear	table
mdy	remove	tables
min	rename	temp
mode	repair	then
modify	report	time
money	required	to
month	resource	today
need	reverse	top
nextfield	revoke	total
noentry	right	trailer
not	rollback	type
noupdate	rollforward	union
null	row	unique
of	rowid	unload
on	screen	unlock
option	select	update
or	serial	upshift
order	set	using
outer	share	values
output	skip	variable
page	smallfloat	verify
pageno	smallint	view
param	some	weekday
pause	space	where
percent	spaces	while
picture	start	with
pipe	statistics	without
print	status	work
printer	step	year
privileges	sum	zerofill
prompt		

Index

This index covers both the User Guide and the Reference Manual. Page numbers that end in U can be found in the User Guide, while those that end in R are in the Reference Manual.

Access privileges

- database privileges 2-102R
- for a synonym 2-77R
- GRANT Introduction-20R
- on views 2-50R
- removing 2-127R
- REVOKE Introduction-20R
- summary of 2-21R
- table privileges 2-101R
- using UNIX permissions for a database 2-72R

ACE

- addition 8-43U
- AFTER GROUP OF control block 8-41U, 5-49R
- aggregates 8-43U, 5-83R
- arithmetic 8-43U
- ASCII expression 5-86R
- AVERAGE aggregate 8-48U
- basic steps for report creation 8-7U
- BEFORE GROUP OF control block 8-41U, 5-52R
- BOTTOM MARGIN statement 8-32U, 5-34R
- calculations 8-43U
- clauses in 8-26U, 5-16R
- CLIPPED expression 5-88R
- COLUMN expression 5-90R
- column headings 8-35U
- command line options 5-9R
- comments 8-26U
- compiling a report specification 8-16U, 8-23U, 5-5R
- compound statement 5-65R
- control block 8-26U, 8-33U, 5-47R
- correcting a report specification 8-23U
- COUNT aggregate 8-48U, 8-51U
- creating a report specification 8-14U
- custom report 8-33U
- DATABASE section 8-11U, 5-16R, 5-17R
- DATE() function 5-93R
- DATE expression 5-92R
- DATE expression, formatting 5-104R
- DAY() function 5-94R
- default report 8-32U
- DEFINE section 5-16R, 5-18R

ACE (Continued)

- definition of 1-29U, 8-6U
- displaying a report 8-57U
- division 8-43U
- dollar sign 8-47U
- error messages 8-24U
- EVERY ROW statement 8-32U, 5-43R
- expressions 5-12R
- expressions, formatting 5-102R, 5-104R
- fill characters 8-46U
- FIRST PAGE HEADER control block 5-55R
- FORMAT section 8-12U, 8-28U, 8-32U, 5-16R, 5-41R
- formatting a report 8-31U
- formatting DATE expressions 5-104R
- formatting numbers 8-45U
- formatting numeric expressions 5-102R
- FOR statement 5-66R
- generating a default report specification 8-15U
- GROUP aggregate 8-54U
- grouping data 8-40U, 5-48R
- headers 8-33U
- IF THEN ELSE statement 5-68R
- INPUT section 5-16R, 5-24R
- joins 8-30U
- keyword 8-26U
- LEFT MARGIN statement 8-32U, 5-30R
- LET statement 5-70R
- LINENO expression 5-95R
- MAX aggregate 8-48U
- MDY() function 5-96R
- MIN aggregate 8-48U
- minimal report 5-16R
- MONTH() function 5-97R
- multiplication 8-43U
- NEED statement 5-72R
- numeric expressions, formatting 5-102R
- ON EVERY ROW control block 8-37U, 5-57R
- ON LAST ROW control block 5-59R
- OUTPUT section 8-31U, 8-57U, 5-16R, 5-27R
- overview 5-5R
- PAGE HEADER control block 8-34U, 5-61R
- PAGE LENGTH statement 8-32U, 5-35R
- PAGENO expression 8-36U, 5-98R
- page numbers 8-36U
- PAGE TRAILER control block 5-63R
- PARAM statement 5-19R

ACE (Continued)

- parentheses 8-45U
- PAUSE statement 5-73R
- PRINT FILE statement 5-78R
- PRINT statement 5-75R
- PRINT USING statement 8-45U
- PROMPT FOR statement 5-25R
- report specification 5-16R
- REPORT TO filename statement 8-58U
- REPORT TO statement 5-28R
- required sections 5-16R
- RIGHT MARGIN statement 5-31R
- running a report 8-17U, 5-5R
- sample specification 8-10U
- sections 5-16R
- selecting all rows and columns 8-29U
- selecting data for a report 8-28U
- selecting data from several tables 8-30U
- SELECT section 8-11U, 8-28U, 5-16R, 5-37R
- SKIP statement 5-79R
- SKIP TO TOP OF PAGE statement 5-80R
- SPACES expression 5-99R
- statement 5-65R
- subtraction 8-43U
- summary of sections 8-26U, 5-16R
- TIME expression 5-100R
- TODAY expression 5-101R
- TOP MARGIN statement 8-32U, 5-33R
- TOTAL aggregate 8-48U
- trailers 8-33U
- USING expression 8-45U, 5-102R
- using with menus 5-5R
- VARIABLE statement 5-21R
- WEEKDAY() function 5-110R
- WHILE statement 5-82R
- YEAR() function 5-111R
- Active table
 - definition of 3-8U, 5-7U
 - selecting a different 5-7U
- Adding data
 - PERFORM 3-16U
 - with RDSQL 7-54U
- ADD keyword, ALTER TABLE statement 2-60R
- ADD NUMERIC Menu
 - Schema editor 2-21U

- Add option
 - ALTER TABLE Menu 2-35U
 - CREATE TABLE Menu 2-17U, 2-19U
 - with PERFORM 4-25R
- ADD STARTING NUMBER Screen
 - schema editor 2-25U
- ADD TYPE Menu 2-20U
- ADD TYPE Menu
 - schema editor 2-24U
- AFTER GROUP OF control block
 - example of use 8-14U
- Aggregate function
 - AVG 2-181R
 - COUNT 2-181R
 - FORMBUILD 3-93R
 - in a SELECT statement 2-181R
 - in SELECT statements 7-35U
 - MAX 2-181R
 - MIN 2-182R
 - RDSQL 7-35U
 - SUM 2-181R
 - with DISTINCT 2-182R
 - with NULL values 2-41R
- ALL keyword, GRANT statement 2-102R
- ALTER INDEX statement
 - definition of 2-16R
 - syntax 2-57R
- Alter option
 - TABLE Menu 2-16U, 2-35U, 1-21R
- ALTER TABLE Menu
 - Build-new-table 2-35U
 - drop option 9-18U
 - Modify option 9-17U
 - options 2-35U
- ALTER TABLE statement
 - ADD 9-11U
 - BEFORE keyword 9-12U
 - combining keywords 9-16U
 - definition of 9-11U, 2-15R, 2-60R
 - DROP keyword 9-13U
 - guidelines for using 2-61R
 - MODIFY keyword 9-14U
- AND keyword, WHERE clause 7-41U
- Append-file option, OUTPUT Menu 7-53U
- Arithmetic operators, in SELECT statements 7-34U

- Asterisk (*) notation
 - in a SELECT statement 7-31U, 2-151R
- Attribute
 - definition of 6-24U
 - how to enter 6-25U
 - keywords 6-24U
 - syntax in FORMBUILD 3-31R
- ATTRIBUTES section
 - FORMBUILD 3-10R, 3-21R
 - form specification file 6-19U
- Audit trail
 - definition of 2-18R
- Audit trails
 - RDSQL 2-119R
- Audit trails, RDSQL 2-69R, 2-92R
- Auto-Indexing 9-30U
- Auto-Indexing
 - definition of 2-26R
- AUTONEXT attribute, FORMBUILD 3-32R
- AVG aggregate function
 - in a SELECT statement 7-36U, 2-181R
 - in FORMBUILD 3-93R
- BCHECK utility, RDSQL 2-65R, 2-125R
- BEFORE keyword, ALTER TABLE statement 2-60R
- BEGIN WORK statement
 - definition of 2-19R
 - syntax and notes 2-63R
- Boolean expressions
 - with NULL values 2-42R
- CHANGE ANYWAY Menu
 - schema editor 9-17U
- CHAR data type
 - acceptable values 2-80R
 - definition of 2-21U
 - determining length 2-21U
 - of a column 9-22U
 - subscripting columns 2-147R
 - with database columns 2-11R
 - with numeric values 9-22U
- CHECK TABLE statement
 - definition of 2-19R, 2-65R
- CHOOSE DATABASE Screen
 - selecting options from 2-13U
- Choose option
 - RDSQL Menu 1-18R

- CLOSE DATABASE statement
 - definition of 9-8U, 2-14R
 - syntax and notes 2-67R
- Clustered Indexes 7-23U
- Clustered Indexes
 - definition of 9-33U, 2-26R
- Clustering
 - ALTER INDEX statement 2-57R
 - CREATE INDEX statement 2-74R
- Column
 - changing column values 2-140R
- Columns
 - adding 2-60R
 - adding, using the ALTER TABLE Menu 9-11U
 - adding, using the TABLE Menu 9-16U
 - changing data type 2-60R
 - changing data types 9-14U
 - column names 1-10U
 - data types 9-22U
 - defining a column 2-18U
 - definition of 1-9U
 - designated as NOT NULL 2-81R
 - determining column length 9-27U
 - dropping, using RDSQL statements 9-13U
 - dropping, using the ALTER TABLE Menu 9-18U
 - indexing 2-26U
 - join 1-13U
 - modifying 2-60R
 - multiple-column indexes 9-31U
 - naming conventions 2-20U, 2-11R, 2-81R, Introduction-10R
 - NULL value in 2-28U, 2-40R
 - removing 2-60R
 - renaming 2-121R
 - renaming, using RDSQL statements 9-11U
 - searching for 1-12U
 - searching for columns within rows 1-12U
 - single-column indexes 9-31U
 - virtual 2-48R, 2-86R
 - with NULL values 9-12U
- Command files
 - in the stores database Introduction-17R
 - naming conventions Introduction-10R
 - saving statements 7-20U
- Command line, operating system
 - accessing ACE 5-9R

- Comments
 - displaying on screen in PERFORM 6-26U
 - in a report specification 8-26U
 - in RDSQL statements 7-29U
- COMMENTS attribute
 - FORMBUILD 3-34R
 - PERFORM 6-26U
- COMMIT WORK statement
 - definition of 9-19U, 2-19R
 - syntax and notes 2-68R
- Compile option
 - FORM Menu 1-16R
 - REPORT Menu 1-20R
- COMPILE REPORT Menu 8-24U
- Compiling
 - a form specification 3-6R
 - a report specification 8-23U, 5-5R
 - definition of 6-11U
- Composite index
 - guidelines for 2-24R
- Composite join, FORMBUILD 3-64R
- Compound statement, ACE 5-65R
- Concurrency control
 - definition of 2-36R
- CONNECT access privilege 2-21R, 2-103R
- Control block
 - ACE 8-26U, 8-33U, 5-47R
 - FORMBUILD 3-71R
- Conventions
 - command file names Introduction-10R
 - file extensions Introduction-11R
 - form names Introduction-10R
 - identifiers Introduction-10R
 - report names Introduction-10R
 - statement syntax Introduction-8R
- Conversion, from prior INFORMIX-SQL releases 2-14U
- COUNT aggregate function
 - in a SELECT statement 7-35U, 2-181R
 - in FORMBUILD 3-93R
- CREATE AUDIT statement 2-69R
- CREATE AUDIT statement
 - definition of 2-18R
- Create Database Screen 2-10U
- CREATE DATABASE statement
 - current database 2-71R
 - definition of 2-14R, 2-71R

- CREATE DATABASE statement (Continued)
 - system catalogs 2-71R
 - WITH LOG IN 2-28R, 2-72R
- CREATE INDEX statement
 - definition of 7-21U, 2-16R
 - syntax 2-74R
 - with ASC 2-75R
 - with DESC 2-75R
 - with UNIQUE and DISTINCT 2-74R
- Create option
 - DATABASE Menu 2-9U, 1-13R
 - TABLE Menu 2-16U, 1-21R
- CREATE SYNONYM statement
 - definition of 2-15R, 2-77R
 - guidelines for using 2-77R
- CREATE TABLE Menu
 - building a schema with 1-25U, 2-17U
 - correcting mistakes 2-18U
 - definition of 2-17U
 - leaving the menu 2-33U
- CREATE TABLE statement
 - definition of 7-11U, 2-15R
 - guidelines for using 2-81R
 - NOT NULL clause 2-40R, 2-81R
 - syntax 2-79R
 - TEMP 2-81R
- CREATE VIEW statement
 - definition of 2-15R, 2-47R
 - guidelines for using 2-85R
 - syntax 2-85R
 - WITH CHECK OPTION 9-21U, 2-49R, 2-87R
- Current database
 - changing 2-13U
 - CLOSE DATABASE statement 2-67R
 - closing 2-67R
 - creating 2-71R
 - DATABASE statement 2-88R
 - definition of 2-11U, 2-14R
 - selecting 2-88R
- current list
 - definition of 4-10U
- Current list
 - PERFORM 3-71R
 - with multiple-table forms 5-20U
- Current option
 - changing 1-20U

- Current option (Continued)
 - definition of 1-19U
 - PERFORM 4-27R
- Current statement, definition of 7-7U
- Current statements
 - definition of 2-8R
- Cursor movement, in PERFORM 6-23U
- customer table
 - columns in 1-10U
 - stores database Introduction-13R
- Customized screen form, FORMBUILD 3-6R, 3-8R
- Data
 - definition of 1-7U
- Database
 - access privileges to 2-21R
 - changing database structure 9-6U
 - changing the current 2-13U
 - closing 2-67R
 - converting from prior INFORMIX-SQL versions 2-14U
 - creating 2-79R
 - creating tables in 2-15U
 - creating through the Main Menu 2-9U
 - current 2-11U
 - database subdirectory 2-10R, 2-71R
 - data manipulation statements 2-16R
 - definition of 1-7U
 - maximum number of open tables Introduction-18R
 - mydata 2-15U
 - naming conventions 2-10U, 2-11R, 2-72R, Introduction-10R
 - recovering 2-29R, 2-30R
 - relational 1-13U, 2-10R
 - removing 2-93R
 - removing, using RDSQL statements 9-8U
 - removing, using the DATABASE Menu 9-9U
 - retrieving information (querying) 1-11U
 - stores demonstration Introduction-13R, Preface-17U
 - system catalogs 2-10R
 - system files in 2-12U
 - tables in 2-10R
 - transactions 2-18R
 - user-menu for 1-34U, 6-5R
 - view of 9-21U
- Database administrator (DBA) access privilege 2-21R, 2-103R
- Database management system
 - definition of 1-6U, Introduction-5R
 - relational 1-13U

DATABASE Menu

guidelines for using 1-13R

options 2-9U, 1-12R

DATABASE section

ACE 5-17R

FORMBUILD 3-10R, 3-12R

in form specifications 6-18U

of report specification 8-11U

DATABASE statement

definition of 2-14R

EXCLUSIVE 2-31R, 2-88R

selecting the current database 2-88R

syntax and notes 2-88R

Data conversion

in an INSERT statement 2-110R

in an UPDATE statement 2-142R

numeric column to CHAR 9-15U

one numeric data type to another 9-14U

Data manipulation statement

DELETE 2-16R

INSERT 2-16R

SELECT 2-16R

UPDATE 2-16R

Data manipulation statements

DELETE 2-90R

INSERT 2-108R

SELECT 2-132R, 2-145R

UPDATE 2-140R

Data type

changing 9-14U

CHAR 2-21U, 9-22U, 2-11R

DATE 2-23U, 9-26U, 2-13R

DECIMAL 9-24U, 2-12R

definition of 4-14R

FLOAT 2-22U, 9-25U, 2-12R

floating decimal point in 2-12R

INTEGER 2-22U, 9-24U, 2-11R

MONEY 2-13R

of columns 9-22U, 2-11R

SERIAL 2-23U, 9-25U, 2-13R

SMALLFLOAT 2-22U, 9-25U

SMALLINT 2-22U, 9-24U, 2-11R

summary 2-20U

summary of numeric 9-23U

table of 2-23U

- Data types
 - changing 2-60R
 - of view columns 2-86R
 - space requirements 2-82R
- Data validation
 - using views 2-50R
- DATE data type
 - acceptable values 2-81R
 - default=today in form specifications 6-29U
 - definition of 2-23U
 - explanation of 2-23U, 9-26U
 - format for display fields 2-82R
 - of a column 9-26U
 - with database columns 2-13R
- Date function
 - in a SELECT statement 7-38U
- DATE function
 - in a SELECT statement 2-183R
- Date function
 - with RDSQL 7-38U
- DAY function
 - in a SELECT statement 7-38U, 2-184R
- DBDELIMITER environment variable Introduction-19R
- DBEDIT environment variable Introduction-20R
- DBPATH environment variable Introduction-20R
- DBPRINT environment variable
 - definition of Introduction-20R
- dbschema
 - definition of 1-25U, 2-72R
- DBTEMP environment variable Introduction-20R
- DBUPDATE utility 2-41R
- DECIMAL data type
 - acceptable values 2-80R
 - definition of 2-22U
 - determining column length 9-27U
 - floating decimal point 2-12R
 - of a column 9-24U
 - scale and precision 2-22U, 9-24U
 - with database columns 2-12R
- DEFAULT attribute
 - FORMBUILD 3-36R
 - PERFORM 6-28U
- DEFINE section, ACE 5-18R
- DELETE keyword
 - GRANT statement 2-101R

- DELETE statement
 - deleting data with 7-59U
 - syntax 2-90R
 - with WHERE clauses 7-59U
- Deleting data
 - PERFORM 3-26U
 - RDSQL 7-59U
 - with DELETE statements 7-59U
- Delimiters
 - definition of 3-7U
 - FORMBUILD 3-15R
- Demonstration database
 - copying Preface-17U
 - customer screen form Introduction-16R
 - description of Introduction-13R
 - introduction to Preface-17U
 - orderform screen form Introduction-16R
 - RDSQL command file Introduction-17R
 - report Introduction-16R
 - restoring the original database Preface-18U
 - sample screen form Introduction-16R
 - screen forms in Introduction-16R
 - tables in 1-8U, Introduction-13R
- Detail option
 - PERFORM 4-28R
- Display-only field, FORMBUILD 3-16R, 3-21R, 3-25R
- Display field, FORMBUILD 3-13R, 3-15R
- display label
 - SELECT clause 2-151R
- Dominant column
 - FORMBUILD 3-29R
 - joins 3-29R
- DOWNSHIFT attribute
 - FORMBUILD 3-38R
 - PERFORM 6-27U
- DROP AUDIT statement 2-92R
- DROP AUDIT statement
 - definition of 2-18R
- DROP DATABASE statement
 - definition of 9-8U, 2-15R
 - syntax and notes 2-93R
- DROP INDEX statement
 - definition of 9-7U, 2-16R
 - syntax and notes 2-95R
- DROP keyword, ALTER TABLE statement 2-61R

- Drop option
 - ALTER TABLE Menu 2-35U, 9-18U
 - CREATE TABLE Menu 2-17U
 - DATABASE Menu 2-9U, 1-13R
 - FORM Menu 1-15R
 - RDSQL Menu 1-18R
 - REPORT Menu 1-19R
 - TABLE Menu 2-16U, 1-22R
- DROP SYNONYM statement
 - definition of 2-15R
 - syntax and notes 2-96R
- DROP TABLE statement
 - definition of 9-6U, 2-15R
 - syntax and notes 2-97R
- DROP VIEW statement
 - definition of 2-15R, 2-47R
 - syntax and notes 2-99R
- Duplicate rows, in a view 2-49R
- Editing keys, PERFORM 4-15R
- Environment variables
 - DBDELIMITER 2-112R, 2-137R, Introduction-19R
 - DBEDIT 2-9R, 3-7R, Introduction-20R
 - DBPATH Introduction-20R
 - DBPRINT Introduction-20R
 - DBTEMP Introduction-20R
 - definition of Introduction-19R
 - INFORMIXDIR Introduction-20R
- EVERY ROW statement, ACE 5-43R
- EXCLUSIVE keyword, LOCK TABLE statement 2-115R
- Exit option
 - ALTER TABLE Menu 2-35U
 - CREATE TABLE Menu 2-17U, 2-33U
 - DATABASE Menu 2-9U
 - FORM Menu 1-15R
 - leaving a menu 1-6R
 - Main Menu 1-14R
 - PERFORM 4-30R
 - RDSQL Menu 1-18R
 - REPORT Menu 1-19R
 - RUN Menu 7-32U
 - TABLE Menu 2-16U, 1-21R
 - USER-MENU Menu 1-23R
- Expression
 - ACE 5-12R
 - in a SELECT statement 2-147R
 - NULL values 2-41R

Extensions

- file Introduction-11R

Field editing keys, PERFORM 4-15R

Fields

- correspondence to columns 1-26U, 3-7U

- definition of 1-26U, 3-6U

- FORMBUILD 3-15R

- join 5-6U

Field tag

- assignments 6-23U

- definition of 6-18U

- FORMBUILD 3-15R

Field width

- explanation of 6-23U

- FORMBUILD 3-15R

File extensions

- .ace Introduction-11R

- .arc Introduction-11R

- .dat 2-34U, 2-10R, Introduction-11R

- .dbs Introduction-11R

- .frm Introduction-11R

- .idx 2-34U, 2-10R, Introduction-11R

- .per Introduction-11R

- .sql Introduction-11R

- definition of Introduction-11R

Filename extensions

- .dbs 2-10R

Fill characters, definition of 8-46U

FLOAT data type

- acceptable values 2-80R

- definition of 2-22U

- of a column 9-25U

- with database columns 2-12R

FORMAT attribute, FORMBUILD 3-39R

FORMAT section, ACE 5-41R

FORMAT section of report specification

- AFTER GROUP OF control block 8-14U

- ON EVERY ROW control block 8-13U

- ON LAST ROW control block 8-14U

- PAGE HEADER control block 8-13U

- PAGE TRAILER control block 8-14U

- sample section 8-12U

Format string, definition of 8-46U

Formatting

- ACE 5-102R, 5-104R

- report 8-31U

FORMBUILD

- ABORT action 3-90R
- action syntax 3-89R
- ADD 3-74R, 3-79R
- AFTER control block 3-74R
- aggregate functions 3-93R
- ALLOWING INPUT 3-25R
- ATTRIBUTES section 3-10R, 3-21R
- attribute syntax 3-31R
- AUTONEXT attribute 3-32R
- AVERAGE (AVG) 3-93R
- BEFORE control block 3-72R
- BEGIN keyword 3-100R
- BELL 3-97R
- COMMENTS action 3-97R
- COMMENTS attribute 3-34R
- composite join 3-64R
- COMPOSITES 3-64R
- control block 3-71R
- COUNT 3-93R
- current list 3-71R
- database column fields 3-23R
- DATABASE keyword 3-12R
- DATABASE section 3-10R, 3-12R
- DATE format 3-40R
- DEFAULT attribute 3-36R
- default form specification file 3-6R, 3-8R
- delimiters 3-15R
- DELIMITERS 3-66R
- DISPLAY 3-74R, 3-87R
- display-only field 3-16R, 3-21R, 3-25R, 3-52R
- display field 3-13R, 3-15R
- display field order 3-21R
- DISPLAYONLY keyword 3-25R
- displaytable 3-26R
- dominant column 3-29R
- DOWNSHIFT attribute 3-38R
- EDITADD 3-72R, 3-74R, 3-76R
- EDITUPDATE 3-72R, 3-74R, 3-76R
- END keyword 3-13R, 3-100R
- EXITNOW 3-95R
- field 3-15R
- field tag 3-15R
- field width 3-15R
- FLOAT format 3-39R
- FORMAT attribute 3-39R

FORMBUILD (Continued)

- IF-THEN-ELSE action 3-99R
- INCLUDE attribute 3-42R
- INSTRUCTIONS section 3-10R, 3-63R
- join 3-28R
- joining columns 3-28R
- LET action 3-92R
- link statement 3-21R
- LOOKUP attribute 3-44R
- lookup fields 3-23R
- lookup join 3-44R
- master-detail relationship 3-68R
- MATCHES 3-100R
- MAX 3-93R
- MIN 3-93R
- NEXTFIELD action 3-95R
- NOENTRY attribute 3-46R
- NOUPDATE attribute 3-48R
- Page layout in SCREEN section 3-13R
- PICTURE attribute 3-50R
- QUERY 3-74R, 3-83R
- QUERYCLEAR attribute 3-52R
- REMOVE 3-74R, 3-85R
- REQUIRED attribute 3-54R
- REVERSE keyword 3-56R, 3-97R
- RIGHT attribute 3-57R
- sample form specification file 3-101R
- SCREEN keyword 3-13R
- SCREEN section 3-10R, 3-13R
- sformbld 3-8R
- SMALLFLOAT format 3-39R
- subscripting a CHAR column 3-16R, 3-24R
- TABLES keyword 3-19R
- TABLES section 3-10R, 3-19R
- TODAY 3-93R
- TOTAL 3-93R
- UPDATE 3-74R, 3-81R
- UPSHIFT attribute 3-58R
- VERIFY attribute 3-60R
- verify join 3-29R
- ZEROFILL attribute 3-62R

FORM Menu

- guidelines for using 1-16R
- options 1-15R

Forms

- multiple-page 4-22U

Forms (Continued)

- naming conventions Introduction-10R

- screen form 1-26U, 3-6U

Form specifications

- ATTRIBUTES section 6-19U

- changing default values 6-28U

- choosing tables for 6-10U

- compiling 6-11U, 1-16R

- controlling case of input 6-27U

- correcting errors in 6-12U

- DATABASE section 6-18U

- default 6-6U, 6-11U, 3-6R, 3-8R

- definition of 6-6U, 3-5R

- field tag assignments 6-23U

- field tags in 6-18U

- files that make up 6-13U

- INSTRUCTIONS section 6-40U

- joining fields in 6-33U

- JOINING keyword in 6-38U

- lookup joins on 6-38U

- maximum number of tables in 6-10U

- modifying 6-7U, 6-15U

- naming 6-9U

- SCREEN section 6-18U

- sections in 6-16U

- specifying acceptable values in 6-31U

- structure of 3-10R

- TABLES section in 6-19U

- using the FORM Menu 1-15R

- using today as default date 6-29U

- verifying input in 6-33U

- verify joins with 6-37U

FROM clause

- in multi-table queries 7-46U

- in RDSQL 7-31U

- OUTER keyword 2-52R, 2-153R

- with SELECT 2-153R

Generate option

- FORM Menu 1-15R

- REPORT Menu 1-19R

- GENERATE REPORT Screen 8-15U

GRANT statement

- CONNECT 2-103R

- DBA 2-103R

- definition of 2-17R

- guidelines for using 2-103R

- GRANT statement (Continued)
 - RESOURCE 2-103R
 - syntax 2-101R
- GROUP BY clause
 - syntax and notes 2-171R
 - with NULL values 2-44R
- HAVING clause, of a SELECT statement 2-173R
- HELP Menu
 - guidelines for using 1-23U
 - options of 1-23U
 - requesting 1-6R, 1-8R
- Highest value operator, PERFORM 4-18U, 4-40R
- Identifiers
 - column Introduction-10R
 - database Introduction-10R
 - menu names 6-19R
 - naming conventions Introduction-10R
 - RDSQL 2-11R
 - table Introduction-10R
- INCLUDE attribute
 - FORMBUILD 3-42R
 - PERFORM 6-31U
- Indexes
 - ascending 2-27U, 7-24U, 9-33U
 - Clustered 2-28U
 - composite 2-24R
 - descending 2-27U, 7-24U, 9-33U
 - for a single column 9-31U
 - for columns with duplicate values 2-27U, 9-32U
 - for multiple columns 9-31U
 - guidelines for creating 9-30U
 - maximum length 9-32U
 - naming conventions 7-21U
 - NULL value 2-41R
 - overview 9-29U
 - removing from a database 9-7U, 2-95R
 - unique 2-27U, 7-23U, 9-32U
 - when to index a table 2-27U, 9-32U, 2-23R
 - with join columns 9-30U
 - with search columns 2-27U, 9-30U
 - with sort columns 2-27U, 9-30U
- INDEX keyword
 - GRANT statement 2-101R
- INFO Menu
 - options 7-61U
 - with RDSQL 7-60U

- Info option
 - RDSQL Menu 1-18R, 2-107R
 - TABLE Menu 2-16U, 1-21R, 2-107R
- Information Lines
 - PERFORM 3-12U
- INFORMIX-SQL Main Menu
 - accessing 2-8U
- INFORMIXDIR environment variable Introduction-20R
- INFO statement 2-20R, 2-106R
- INPUT section, ACE 5-24R
- INSERT keyword
 - GRANT statement 2-101R
- INSERT statement
 - adding data 7-54U
 - format for 7-54U
 - guidelines for using 2-109R
 - inserting through a view 2-49R
 - RDSQL 7-54U
 - SERIAL columns in 2-109R
 - syntax 2-108R
 - VALUES clauses with 7-54U
 - WHERE clauses with 7-55U
 - with NULL values 2-44R
- INSTRUCTIONS section
 - FORMBUILD 3-10R, 3-63R
 - in form specification 6-40U
- INTEGER data type
 - acceptable values 2-80R
 - definition of 2-22U
 - of a column 9-24U
 - with database columns 2-11R
- INTO TEMP clause
 - INSERT statement 2-109R
 - SELECT statement 2-177R
- IS NULL keywords 2-43R
- items table
 - stores database Introduction-14R
- Join
 - column, definition of 5-11U
 - columns, FORMBUILD 3-28R
 - columns, naming in SELECT clauses 7-46U
 - columns, naming in WHERE clauses 7-46U
 - columns in SELECT statements 7-46U
 - definition of 1-13U
 - dominant column 3-29R
 - field, definition of 5-11U

Join (Continued)

- fields 5-6U, 6-33U
- fields on screen forms 5-11U
- FORMBUILD 3-28R
- in a SELECT statement 2-164R
- indexing join columns 9-30U
- in PERFORM 5-11U
- in the stores database 1-13U
- joined columns with the same name 2-165R
- JOINING keyword 6-38U
- multiple-table queries 7-46U
- multiple joins 2-165R
- outer join 2-52R, 2-165R
- self-join 2-165R
- syntax 7-46U
- where a column is NULL 2-43R
- JOINING keyword in a form specification 6-38U

Keys

- basic keys for SQL Preface-15U
- CTRL-C key 1-8R
- cursor positioning 3-21U, 4-16R
- delete key 1-8R
- editing keys in PERFORM 3-23U
- editing keys in RDSQL 7-12U
- for displaying HELP Menu 1-23U
- for leaving a menu 2-14U
- for moving the highlight 1-20U
- for requesting help 1-6R, 1-8R
- selecting options from a menu 1-6R
- Link statement, FORMBUILD 3-21R
- LOAD statement 2-19R, 2-112R
- LOCK TABLE statement
 - definition of 2-17R
 - EXCLUSIVE 2-115R
 - MODE 2-115R
 - SHARE 2-115R
 - syntax and notes 2-115R
 - with PERFORM 4-25R
- LOOKUP attribute, FORMBUILD 3-44R
- Lookup fields
 - definition of 5-15U, 6-38U
 - FORMBUILD 3-23R
 - in form specifications 6-38U
- Lookup joins
 - definition of 6-38U
 - in PERFORM 5-15U

- Lookup joins (Continued)
 - on a screen form 5-15U
- Lowest value operator, PERFORM 4-18U, 4-40R
- Main Menu
 - accessing 1-17U, 2-8U, 1-5R
 - Database option 1-21U, 2-9U, 1-12R
 - Exit option 1-21U, 1-14R
 - Form option 1-19U, 1-21U, 6-8U, 1-15R
 - map of INFORMIX-SQL menu hierarchy 1-8R
 - Query-language option 1-21U, 1-17R
 - Report option 1-21U, 1-19R
 - Table option 1-21U, 1-21R
 - User-menu option 1-21U, 1-23R, 6-6R
- manufact table
 - stores database Introduction-14R
- Master-detail relationship
 - introduction to 5-6U, 5-16U
 - PERFORM 3-68R
- Master option
 - PERFORM 4-31R
- MATCHES keyword
 - how to use 7-42U
 - with WHERE clauses 7-41U
 - with wildcard characters 7-42U
- Math functions
 - display labels 7-37U
 - examples 7-37U
 - in 7-35U
- MAX aggregate function
 - in a SELECT statement 7-36U, 2-181R
 - in FORMBUILD 3-93R
- MDY function
 - in a SELECT statement 7-38U, 2-185R
- menuform screen form
 - definition of 6-10R
 - entering data into 6-10R
 - menu name field 6-19R
 - menu title field 6-20R
 - selection action field 6-26R
 - selection number field 6-21R
 - selection text field 6-25R
 - selection type 6-23R
- Menus
 - ADD TYPE 2-20U
 - ALTER TABLE 2-35U
 - CHANGE ANYWAY 9-17U

Menus (Continued)

- CREATE TABLE 2-17U
- current option 1-19U
- custom (user-menus) 1-34U, 6-5R
- DATABASE 2-9U
- entering text 1-7R
- HELP 1-23U
- INFORMIX-SQL Main Menu 1-17U, 2-8U
- parts of a menu screen 1-5R
- parts of text-entry screens 1-7R
- REMOVE 9-18U
- REPORT 8-8U
- selecting options 1-20U, 1-6R
- TABLE 2-15U
- Message line of a menu 1-19U
- MIN aggregate function
 - in a SELECT statement 7-36U, 2-182R
 - in FORMBUILD 3-93R
- MODE keyword, LOCK TABLE statement 2-115R
- Modifying data
 - with PERFORM 3-25U
 - with RDSQL 7-57U
- MODIFY keyword, ALTER TABLE statement 2-61R
- Modify option
 - ALTER TABLE Menu 2-35U, 9-17U
 - CREATE TABLE Menu 2-17U
 - FORM Menu 1-15R
 - RDSQL Menu 1-17R
 - REPORT Menu 1-19R
 - USER-MENU Menu 1-23R
- MODIFY REPORT Menu
 - Compile option 8-21U
 - Discard-and-exit option 8-22U
 - Save-and-Exit option 8-21U
- MODIFY REPORT Screen 8-20U
- MONEY data type
 - acceptable values 2-81R
 - definition of 2-24U
 - determining column length 2-24U, 9-27U
 - explanation of 9-26U
 - with database columns 2-13R
- MONTH function
 - in a SELECT statement 7-38U, 2-186R
- Multiple-table form, definition of 5-6U
- Multiple-table queries
 - FROM clauses 7-46U

- Multiple-table queries (Continued)
 - with RDSQL 7-46U
- Multiusers systems
 - Privileges Introduction-20R
- mydata database 2-15U
- Naming conventions
 - columns 2-20U, 2-11R, 2-81R, Introduction-10R
 - database 2-10U, 2-11R, 2-72R
 - for report specifications 8-15U
 - identifiers Introduction-10R
 - restrictions 2-11U
 - table 2-16U, 2-11R, 2-81R
 - user-menu names 6-19R
- Nested sorts, with ORDER BY clauses 7-45U
- New-file option, OUTPUT Menu 7-52U
- New option
 - FORM Menu 1-15R
 - RDSQL Menu 1-17R
 - REPORT Menu 1-19R
- Next option
 - with PERFORM 4-32R
 - with RDSQL 7-32U
- NOENTRY attribute, FORMBUILD 3-46R
- NOUPDATE attribute, FORMBUILD 3-48R
- NULL values
 - assigned to a column 9-12U
 - definition of 2-40R
- Null values
 - in a column 2-28U
- NULL values
 - in a column 2-40R
 - in a GROUP BY clause 2-44R
 - in an ORDER BY clause 2-44R
 - in Boolean expressions 2-42R
 - in expressions 2-41R
 - in joined columns 2-43R
 - NOT NULL clause 2-40R, 2-81R
 - truth tables 2-42R
 - with INSERT 2-44R
 - with UPDATE 2-44R
- ON EVERY ROW control block
 - example of use 8-13U
- ON LAST ROW control block
 - example of use 8-14U
- Operator
 - UNION 2-178R

- Options
 - selecting from a menu 1-6R
- ORDER BY clause
 - ASC 2-175R
 - DESC 2-175R
 - format for 7-43U
 - in an INSERT statement 2-109R
 - in a SELECT statement 2-175R
 - in nested sorts 7-45U
 - with NULL values 2-44R
- orders table
 - stores database Introduction-13R
- OR keyword, WHERE clause 7-41U
- OUTPUT Menu
 - Append-file option 7-53U
 - New-file option 7-52U
 - Printer option 7-51U
- Output option
 - PERFORM 4-33R
 - RDSQL Menu 7-51U, 1-17R
- OUTPUT section, ACE 5-27R
- OUTPUT statement
 - definition of 2-20R
 - RDSQL 2-117R
- PAGE HEADER control block
 - example of use 8-13U
- PAGE TRAILER control block
 - example of use 8-14U
- PERFORM
 - active table 3-21R, 3-22R
 - adding data with 3-16U, 4-14R, 4-25R
 - Add option 3-16U, 4-10R, 4-25R
 - CHAR fields in 3-19U, 4-16U, 4-14R
 - comments line 3-34R
 - control keys in 4-15R
 - current list 4-10U, 3-71R, 4-23R
 - Current option 4-10R, 4-27R
 - cursor positioning keys 3-21U, 4-16R
 - data checking in 4-20R
 - data types 4-14R
 - DATE fields in 3-19U, 4-15U, 4-15R
 - DECIMAL fields in 4-15R
 - Delete option 3-26U
 - deleting data with 3-26U, 4-42R
 - Detail option 5-16U, 4-10R, 4-28R
 - display field order in 4-19R

PERFORM (Continued)

- displaying comments on the screen 6-26U
- editing keys in 3-23U, 4-15R
- equal to operator 4-14U
- example queries 4-21U
- exiting from 3-27U, 4-10R, 4-30R
- Exit option 4-11R, 4-30R
- explanation of queries in 4-13U
- files needed for 4-6R
- FLOAT fields in 4-15R
- highest value operator 4-18U, 4-40R
- how to access 3-9U
- how to call up 4-6R
- Information Lines in 3-12U, 4-8R, 4-10R
- insert mode 3-22U, 4-17R
- INTEGER fields in 4-14R
- introduction to 3-8U, 4-5R
- LOCK statement with 4-25R
- lowest value operator 4-18U, 4-40R
- Master option 5-19U, 4-10R, 4-31R
- menuform screen 6-10R
- menu options 3-14U, 1-5R
- menu ptions 4-10R
- modifying data with 3-25U, 4-46R
- MONEY fields in 3-19U, 4-15U, 4-15R
- multiple-page forms 4-22U
- Next option 4-11U, 4-10R, 4-32R
- numeric fields in 3-18U, 4-15U
- Output option 4-23U, 4-11R, 4-33R
- permissions in 4-21R
- pictures in 3-20U
- Previous option 4-11U, 4-10R, 4-36R
- query operators in 4-38R
- Query option 4-10R, 4-37R
- range operator 4-17U
- relational operators in 4-14U, 4-38R
- Remove option 4-10R, 4-42R
- removing data with 4-42R
- running, from command line 4-13R
- screen forms with 4-11R
- Screen option 4-22U, 4-10R, 4-44R
- search criteria in 4-7U
- SERIAL fields in 3-18U, 4-14R
- SMALLFLOAT fields in 4-15R
- SMALLINT fields in 4-14R
- Status Lines in 3-13U, 4-8R, 4-12R

PERFORM (Continued)

table of query operators 4-13U

Table option 5-7U, 4-10R, 4-45R

typeover mode 3-22U, 4-17R

Update option 3-25U, 4-10R, 4-46R

user access privileges in 4-21R

wildcard character in 4-39R

wildcard characters in 4-16U

Permissions

in RDSQL Introduction-20R

UNIX Introduction-20R

PICTURE attribute

definition of 3-20U

in FORMBUILD 3-50R

Previous option, PERFORM 4-36R

Printer option, OUTPUT Menu 7-51U

Printing query results

with PERFORM 4-33R

with RDSQL 7-51U

Privileges

Multiuser systems Introduction-20R

Program interrupt key

DOS Preface-16U

UNIX Preface-16U

PUBLIC keyword, of a GRANT statement 2-102R

Query

compound queries 2-178R

definition of 1-11U, 4-6U

examples with PERFORM 4-21U

operators, PERFORM 4-38R

option, PERFORM 4-37R

RDSQL Menu 1-17R

results with RDSQL 7-32U

saving results of in PERFORM 4-23U, 4-36R

saving results of with RDSQL 7-51U

searching for columns 1-12U

searching for columns within rows 1-12U

searching for rows 1-11U

searching for rows with NULL values 2-43R

subqueries 2-167R

through a view 2-48R

with RDSQL 7-30U

with SELECT statements 7-30U

QUERYCLEAR attribute, FORMBUILD 3-52R

Querying the database

subqueries 2-141R

Query language

definition of 2-6U

Range operator, in PERFORM 4-17U, 4-39R

RDSQL

ACCESS FOR keywords, INFO statement 2-106R

adding data 7-54U

ALTER INDEX statement 2-57R

ALTER TABLE statement 2-60R

audit trails 2-69R, 2-92R, 2-119R

Auxiliary statements 2-19R

BEGIN WORK statement 2-63R

changing data type of a column 9-14U

CHECK TABLE statement 2-65R, 2-125R

CLOSE DATABASE statement 2-67R

command file 7-21U

commenting 7-29U

COMMIT WORK statement 2-68R

comparison to PERFORM 7-9U

CREATE AUDIT statement 2-69R, 2-119R

CREATE DATABASE statement 2-71R

CREATE INDEX statement 2-74R

CREATE SYNONYM statement 2-77R

CREATE TABLE statement 2-79R

CREATE VIEW statement 2-85R

current statements 7-7U, 2-8R

data access statements 2-17R

DATABASE statement 2-88R

data conversion 2-110R

data definition statements 2-14R

data integrity statements 2-18R

data manipulation statements 2-16R

DELETE statement 2-90R

DROP AUDIT statement 2-92R, 2-119R

DROP DATABASE statement 2-93R

DROP INDEX statement 2-95R

dropping a column 9-13U

dropping an index 9-7U

dropping a table 9-6U

DROP SYNONYM statement 2-96R

DROP TABLE statement 2-97R

DROP VIEW statement 2-99R

editing keys 7-12U, 7-28U

editor 7-12U, 7-14U, 7-19U

entering statements 7-12U, 7-28U

formatting statements 7-29U

FROM clause 7-31U

RDSQL (Continued)

- GRANT statement** 2-101R
- how to use** 2-8R
- identifiers** 2-11R
- INDEXES FOR keywords, INFO statement** 2-106R
- INFO statement** 2-106R
- INSERT INTO keywords, LOAD statement** 2-112R
- INSERT statement** 2-108R
- introduction to** 2-7U
- LOAD statement** 2-112R
- LOCK TABLE statement** 2-115R
- long statements** 7-14U
- modifying a table** 9-11U
- modifying data** 7-54U, 7-57U
- multi-table queries** 7-46U
- naming current statements** 7-20U
- NEW Screen** 7-28U
- Next option** 7-32U
- OUTPUT statement** 2-117R
- PIPE keyword, OUTPUT statement** 2-117R
- PRIVILEGES keywords, INFO statement** 2-107R
- Query-language** 2-8R
- query results** 7-32U
- RECOVER TABLE statement** 2-119R
- removing a database** 9-8U
- RENAME COLUMN statement** 2-121R
- RENAME TABLE statement** 2-123R
- renaming a column** 9-11U
- renaming a table** 9-7U
- REPAIR TABLE statement** 2-125R
- Restart option** 7-32U
- REVOKE statement** 2-127R
- ROLLBACK WORK statement** 2-130R
- ROLLFORWARD DATABASE statement** 2-131R
- RUN Menu** 7-32U
- running statements** 7-18U, 7-28U
- saving statements** 7-20U
- SELECT clause** 7-31U
- SELECT statement** 7-30U, 2-117R, 2-132R, 2-137R, 2-145R
- separating with semicolons** 7-30U
- SET LOCK MODE statement** 2-133R
- specifying a system editor** 7-14U
- START DATABASE statement** 2-135R
- statements** 1-32U, 7-6U, 2-8R, 2-56R
- STATUS FOR keywords, INFO statement** 2-107R
- Structured Query Language (SQL)** 2-7R

- RDSQL (Continued)
 - subscribing CHAR columns 2-147R
 - system editors 7-14U
 - TABLES keyword, INFO statement 2-106R
 - UNLOAD statement 2-137R
 - UNLOCK TABLE statement 2-139R
 - update notes for Version 1.10 users 2-41R
 - UPDATE statement 7-57U, 2-140R
 - UPDATE STATISTICS statement 2-144R
 - Use-editor option 7-14U
 - WITHOUT HEADINGS keywords, OUTPUT statement 2-117R
- RDSQL Menu
 - Choose option 7-8U, 2-9R
 - Drop option 7-8U, 2-9R
 - Exit option 7-8U, 2-9R
 - guidelines for using 1-18R
 - Info option 7-8U, 7-60U, 2-9R
 - introduction 7-28U
 - introduction to 7-7U
 - Modify option 7-8U, 7-19U, 2-9R
 - New option 7-8U, 7-28U, 2-8R
 - options 1-17R
 - Output option 7-8U, 2-9R
 - Run option 7-8U, 7-28U, 2-9R
 - Save option 7-8U, 7-20U, 2-9R
 - Use-editor option 7-8U, 2-9R
- RDSQL statement
 - examples 7-29U
 - formatting 7-29U
- RDSQL statements
 - how to enter 7-12U
 - how to run 7-18U
- RECOVER TABLE statement 2-119R
- RECOVER TABLE statement
 - definition of 2-19R
- Relational DBMS
 - advantages 1-15U
 - definition of 1-13U
- Relational operators
 - in PERFORM 4-14U, 4-38R
 - in RDSQL 7-39U
 - in RDSQL WHERE clauses 7-39U
 - in SELECT statements 7-39U
 - with CHAR fields in PERFORM 4-16U
 - with DATE fields in PERFORM 4-15U
 - with MONEY fields in PERFORM 4-15U

- Relational operators (Continued)
 - with numeric fields in PERFORM 4-15U
- REMOVE Menu 9-18U
- Remove option, PERFORM 4-42R
- RENAME COLUMN statement
 - definition of 9-11U, 2-16R
 - guidelines for using 2-121R
 - syntax 2-121R
- RENAME TABLE statement
 - definition of 9-7U, 2-15R
 - syntax and notes 2-123R
- REPAIR TABLE statement 2-19R, 2-125R
- REPORT Menu
 - Compile option 8-8U
 - Drop option 8-8U
 - Exit option 8-8U
 - Generate option 8-8U, 8-15U
 - guidelines for using 1-20R
 - Modify option 8-8U, 8-20U
 - New option 8-8U
 - Run option 8-8U, 8-18U
 - summary of options 8-8U, 1-19R
- Reports
 - ACE overview 5-5R
 - basic steps for creating 8-7U
 - compiling 1-20R, 5-5R
 - definition of 8-6U
 - in the stores database Introduction-16R
 - minimal 5-16R
 - modifying 8-20U
 - naming conventions 8-15U, Introduction-10R
 - running 8-17U, 5-5R
 - sample report 8-8U
- Report specification
 - compiling 8-23U
 - correcting 8-23U
 - DATABASE section 8-11U
 - default, compiling 8-16U
 - default, creating 8-15U
 - definition of 8-7U
 - FORMAT section 8-12U
 - modifying 8-20U
 - required sections 5-16R
 - SELECT section 8-11U
 - summary of sections 8-26U

- REQUIRED attribute, FORMBUILD 3-54R
- Required sections
 - ACE report 5-16R
- Reserved words, restrictions on 2-11U
- RESOURCE access privilege 2-21R, 2-103R
- Restart option, RDSQL 7-32U
- REVERSE keyword, FORMBUILD 3-56R
- REVOKE statement
 - ALL 2-127R
 - ALTER 2-127R
 - CONNECT 2-128R
 - DBA 2-128R
 - definition of 2-17R
 - DELETE 2-127R
 - guidelines for using 2-128R
 - INDEX 2-127R
 - INSERT 2-127R
 - RESOURCE 2-128R
 - SELECT 2-127R
 - syntax 2-127R
 - UPDATE 2-127R
- RIGHT attribute, FORMBUILD 3-57R
- ROLLBACK WORK statement
 - definition of 2-19R
 - syntax and notes 2-130R
- ROLLFORWARD DATABASE statement
 - definition of 2-19R
 - recovering a database 2-29R, 2-30R
 - syntax 2-131R
- Row
 - adding rows to a table 1-9U
 - calculating row length 9-28U
 - definition of 1-9U, 2-10R
 - deleting from a table 2-90R
 - in the stores database 1-9U
- ROWID 2-53R
 - searching for 1-11U
- ROWID keyword 2-53R
- RUN Menu
 - RDSQL 7-32U
- Run option
 - FORM Menu 1-15R
 - RDSQL Menu 1-17R
 - REPORT Menu 1-19R
 - USER-MENU Menu 1-23R

RUN REPORT Screen 8-18U

Save option

RDSQL Menu 1-18R

Saving query results

with PERFORM 4-33R

with RDSQL 7-51U

Schema, for a table 1-25U, 2-17U

Schema editor

adding a column 9-16U

ADD NUMERIC Menu 2-21U

ADD STARTING NUMBER Screen 2-25U

ADD TYPE Menu 2-24U

ALTER TABLE 9-16U

assigning data types 2-20U

CHANGE ANYWAY Menu 9-17U

changing a table 2-35U

correcting mistakes 2-18U

defining a column 2-18U

definition of 2-18U

exiting 2-32U

EXIT Menu 2-33U

naming a table 2-16U

Screen forms

active table on 5-7U

choosing tables when creating 6-10U

creating multiple 3-9U

default 6-13U

definition of 1-26U, 3-6U

definition of multiple-table 5-6U

in the demonstration database 3-6U, Introduction-16R

join fields on 5-11U

lookup joins on 5-15U

master-detail relationships on 5-6U

maximum number of tables on 3-8U, 6-10U

modifying 6-7U

naming 6-9U

using the FORM Menu 1-15R

verify joins on 5-14U

Screen option

ALTER TABLE Menu 2-35U

CREATE TABLE Menu 2-17U

PERFORM 4-44R

SCREEN section

FORMBUILD 3-10R, 3-13R

Search conditions

in RDSQL WHERE clauses 7-39U

- Search conditions (Continued)
 - in SELECT statements 7-39U
- Search criteria
 - definition of 4-7U
 - in PERFORM 4-7U
- select-list
 - SELECT clause 2-150R
- SELECT clause
 - ALL keyword 2-150R
 - DISTINCT and UNIQUE keywords 2-150R
 - guidelines for using 2-150R
 - in RDSQL 7-31U
 - syntax 2-150R
- SELECT keyword
 - GRANT statement 2-101R
- Select option
 - DATABASE Menu 2-9U, 1-13R
- SELECT section
 - of report specification 8-11U, 5-37R
- SELECT statement
 - aggregate functions in 2-181R
 - arithmetic operators in 7-34U
 - clauses in 7-30U
 - creating temporary tables 2-177R
 - defining a view 2-47R
 - definition of 2-132R
 - display labels in 7-35U
 - examples 7-31U, 7-32U, 7-33U, 7-34U, 7-35U
 - expression in 2-147R
 - FROM clause 2-153R
 - GROUP BY clause 2-171R
 - HAVING clause 2-173R
 - in a report specification 8-11U
 - INSERT statement 2-108R
 - INTO TEMP clause 2-177R
 - joining columns with 7-46U, 2-164R
 - ORDER BY clause 7-43U, 2-175R
 - overview of clauses 2-149R
 - relational operators 7-39U, 2-148R
 - search conditions 7-39U
 - SELECT clause 2-150R
 - UNION operator 2-178R
 - WHERE clause 7-38U, 2-155R
 - with a date function 7-38U
 - with aggregates 7-35U
 - with an outer join 2-52R, 2-165R

- SELECT statement (Continued)
 - with a self-join 2-165R
 - with asterisks 7-31U
 - with AVG 7-36U
 - with COUNT 7-35U
 - with date functions 7-38U
 - with DAY 7-38U
 - with math functions 7-35U
 - with MAX 7-36U
 - with MDY 7-38U
 - with MIN 7-36U
 - with MONTH 7-38U
 - with multiple joins 2-165R
 - with subqueries 2-167R
 - with SUM 7-35U
 - with WEEKDAY 7-38U
 - with YEAR 7-38U
- SERIAL data type
 - acceptable values 2-81R
 - assigning starting number 2-25U
 - definition of 2-23U
- INSERT statement 2-109R
 - of a column 9-25U
 - with database columns 2-13R
- SET clause
 - in an UPDATE statement 7-57U
- SET keyword, UPDATE statement 2-140R
- SET LOCK MODE statement
 - definition of 2-17R
 - syntax 2-133R
- sformbld
 - definition of 3-8R
- SHARE keyword, LOCK TABLE statement 2-115R
- SMALLFLOAT data type
 - acceptable values 2-80R
 - definition of 2-22U
 - of a column 9-25U
 - with database columns 2-12R
- SMALLINT data type
 - acceptable values 2-80R
 - definition of 2-22U
 - of a column 9-24U
 - with database columns 2-11R
- Sorting data
 - with NULL values 2-44R
 - with ORDER BY 7-43U, 2-175R

- Sorting data (Continued)
 - with RDSQL 7-43U
- START DATABASE statement
 - definition of 2-19R
 - syntax and notes 2-135R
- Statements
 - ACE 8-26U, 5-16R, 5-65R
 - definition of 1-32U, 7-6U
 - editing in RDSQL 7-12U
 - RDSQL 2-8R, 2-56R
 - saving in a command file 7-20U
 - syntax conventions Introduction-8R
- Statement type
 - auxiliary 2-19R
 - data access 2-17R
 - data definition 2-14R
 - data integrity 2-18R
 - data manipulation 2-16R
- Status lines
 - about 3-34R
 - PERFORM 3-13U
- stock table
 - stores database Introduction-14R
- stores database
 - copying Preface-17U
 - creating Preface-19U
 - customer screen form Introduction-16R
 - customer table 9-10U
 - joined columns in 1-13U
 - orderform screen form Introduction-16R
 - overview Preface-17U
 - RDSQL command file Introduction-17R
 - report specifications 8-8U, 8-10U, Introduction-16R
 - restoring the original Preface-18U
 - sample screen form Introduction-16R
 - screen forms in Introduction-16R
 - summary of tables 1-8U
 - tables in Introduction-13R
 - user-menu 6-7R
 - user-menu design outline 6-9R
- SUM aggregate function
 - in a SELECT statement 7-35U, 2-181R
- Synonym
 - creating 2-77R
 - for a table 2-153R
 - in a SELECT statement 7-35U

Synonym (Continued)

- removing a synonym 2-96R

Syntax conventions

- statements Introduction-8R

sysmenuitems table

- definition of 6-11R

- sample data in 6-14R

sysmenus table

- definition of 6-11R

- sample data in 6-12R

System catalogs

- creating 2-71R

- definition of 2-12U, 2-10R

- syscolumns 2-47R

- systables 2-144R

- sysviews 2-47R

Table

- access privileges 2-21R

- active in PERFORM 3-8U, 5-7U

- adding a column 2-60R

- alias for table name 2-153R

- changing table structure 2-35U, 9-9U, 2-60R

- creating an alternate name (synonym) 2-77R

- creating a temporary table 2-177R

- creating through menus 1-24U, 2-17U

- defining a column 2-18U

- definition of 1-8U, 2-10R

- guidelines for indexing 2-23R

- in the stores database 1-8U, Introduction-13R

- joining columns 1-13U, 2-164R

- locking 2-115R

- maximum number of open tables Introduction-18R

- modifying a row 2-140R

- naming conventions 2-16U, 2-11R, 2-81R

- outer join 2-52R

- removing, using RDSQL statements 9-6U

- removing, using the TABLE Menu 9-7U

- removing a column 2-60R

- removing a synonym 2-96R

- removing a table 2-97R

- renaming 9-7U, 2-123R

- revoking user access 2-128R

- schema 1-25U, 2-17U

- sysmenuitems 6-11R

- sysmenus 6-11R

- system files that make up a table 2-34U

- Table (Continued)
 - unlocking 2-139R
- TABLE Menu
 - calculating row lengths 9-28U
 - guidelines for using 1-22R
 - options of 2-16U
 - schema editor 9-16U
 - summary of options 1-21R
- Table option, PERFORM 4-45R
- table schema
 - how to reproduce 1-25U
- TABLES keyword, FORMBUILD 3-19R
- TABLES section
 - FORMBUILD 3-10R, 3-19R
 - in form specifications 6-19U
- termcap file Introduction-18R
- Terminal types Introduction-18R
- terminfo file Introduction-18R
- Text-Entry screens
 - definition of 1-7R
- Text-entry screens
 - entering text 1-7R
 - requesting help 1-8R
- Text editors
 - choosing Introduction-19R
 - internal editor Introduction-19R
 - RDSQL editor Introduction-19R
 - system editor Introduction-19R
- TOTAL aggregate function
 - in FORMBUILD 3-93R
- Transaction
 - committing modifications 2-68R
 - COMMIT WORK 9-19U
 - cycle 2-28R
 - definition of 9-19U, 2-18R, 2-28R
 - log 2-28R, 2-29R, 2-72R, 2-135R
 - log file maintenance 2-30R
 - recovering transactions 2-131R
 - starting 2-63R
 - undoing modifications 2-130R
- UNION operator 2-47R
- UNIX file permissions Introduction-20R
- UNLOAD statement 2-20R, 2-137R
- UNLOCK TABLE statement
 - definition of 2-17R
 - syntax and notes 2-139R

- UPDATE keyword
 - GRANT statement 2-102R
- Update option, PERFORM 4-46R
- UPDATE statement
 - data conversion in 2-142R
 - guidelines for using 2-141R
 - in RDSQL 7-57U
 - subqueries 2-141R
 - syntax 2-140R
 - updating through a view 2-48R
 - with NULL values 2-44R
- UPDATE STATISTICS statement
 - definition of 2-16R
 - FOR TABLE 2-144R
 - syntax and notes 2-144R
- UPSHIFT attribute
 - FORMBUILD 3-58R
 - PERFORM 6-27U
- Use-editor option
 - RDSQL Menu 1-17R
- User-menu
 - accessing, through Main Menu 6-6R
 - creating, through menuform 6-10R, 6-16R
 - definition of 1-34U, 6-5R
 - designating a selection number 6-21R
 - designating a selection type 6-23R
 - layout specifications 6-8R
 - levels of submenus 6-8R
 - modifying 6-18R
 - naming conventions 6-19R
 - options 6-25R
 - sample design outline 6-9R
 - sysmenuitems information 6-11R
 - sysmenus information 6-11R
- USER-MENU Menu
 - guidelines for using 1-23R
 - options 1-23R
- User-menu option
 - Main Menu 6-6R
- VALUES keyword, INSERT statement 7-54U, 2-108R
- VERIFY attribute
 - FORMBUILD 3-60R
 - PERFORM 6-33U
- Verify joins
 - definition of 6-37U
 - FORMBUILD 3-29R

- Verify joins (Continued)
 - in form specifications 6-37U
 - on a screen form 5-14U
- View
 - access privileges 2-50R
 - alternate name (synonym) 2-77R
 - characteristics 2-85R
 - creating 2-47R
 - definition of 9-21U, 2-46R
 - deleting 2-47R, 2-99R
 - limitations 2-46R
 - modify the database 2-48R
 - query the database 2-48R
 - virtual column 2-86R
 - with duplicate rows 2-49R
- Wait for lock
 - definition of 2-39R
 - SET LOCK MODE statement 2-133R
- WEEKDAY function
 - in a SELECT statement 7-38U, 2-187R
- WHERE clause
 - ALL keyword 2-168R
 - AND keyword 7-41U
 - ANY keyword 2-168R
 - comparison condition 2-155R
 - DATE function 2-183R
 - DAY function 2-184R
 - EXISTS keyword 2-168R
 - IN 2-158R
 - in DELETE statements 7-59U
 - in INSERT statements 7-55U
 - IN keyword 2-168R
 - in SELECT statements 7-38U
 - in UPDATE statements 7-57U
 - joining columns in 7-46U, 2-164R
 - LIKE keyword 2-159R
 - MATCHES 7-41U
 - MATCHES keyword 2-161R
 - MDY function 2-185R
 - MONTH function 2-186R
 - NULL values 2-163R
 - OR keyword 7-41U
 - pattern matching in 2-161R
 - relational operators 7-39U
 - search conditions 7-39U, 2-146R, 2-155R
 - SOME keyword 2-168R

- WHERE clause (Continued)
 - syntax 2-155R
 - WEEKDAY function 2-187R
 - wildcard characters in 7-42U
 - with a subquery 2-167R
 - with BETWEEN 2-157R
 - with NULL values 2-43R
 - with relational operators 2-156R
 - with UPDATE 2-141R
 - YEAR function 2-188R
- Wildcard character
 - in PERFORM 4-39R
 - in RDSQL WHERE clauses 7-42U
- Wildcard characters
 - in PERFORM 4-16U
- Writing query results to a file
 - PERFORM 4-33R
 - with RDSQL 7-51U
- YEAR function
 - in a SELECT statement 7-38U, 2-188R
- ZEROFILL attribute, FORMBUILD 3-62R