INFORMIX-SQL

Relational Database Management System

REFERENCE MANUAL

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

Part No. 200-404-2020-0
Rev. A
PRINTED IN U.S.A.

INFORMIX-SQL
Version 2.10

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

Introduction

What Is INFORMIX-SQL? 5
About This Manual 6
Conventions 8
  Syntax 8
  Identifiers 10
  Database, Form, Report, and Command Filenames 10
  File Extensions 11
  System Catalogs 12
The Demonstration Database 13
  Tables in the Database 13
  Sample Screen Forms 16
  Sample Reports 16
  Sample RDSQL Command Files 17
Terminal and Operating System Interactions 18
  Number of Open Tables 18
  Terminal Types (UNIX Systems Only) 18
  Text Editors 19
  Environment Variables 19
  User Access Privileges on Multi-User Systems 20

Chapter 1. The INFORMIX-SQL Main Menu

Accessing INFORMIX-SQL on DOS Systems 1-5
  Loading the Database Agent 1-5
  Entering INFORMIX-SQL 1-6
Accessing INFORMIX-SQL on UNIX Systems 1-6
The INFORMIX-SQL Screens 1-6
  Menu Screens 1-7
  Text-Entry Screens 1-8
Maps of the Menu Structure 1-9
The INFORMIX-SQL Main Menu Options 1-11

Chapter 2. The RDSQL Query Language

Overview 2-7
How to Use RDSQL 2-8
Overview of RDSQL 2-10
Formatting RDSQL Statements 2-10
RDSQL Identifiers 2-11
Database Data Types 2-11
RDSQL Statement Summary 2-14
Data Definition Statements 2-14
Data Manipulation Statements 2-16
Data Access Statements 2-17
Data Integrity Statements 2-18
Auxiliary Statements 2-19
User Status and Privileges 2-21
Indexing Strategy 2-23
Auto-Indexing 2-26
Clustered Indexes 2-26
Transactions 2-28
Transaction Log File Maintenance 2-30
Audit Trails 2-32
Comparing Audit Trails and the Transaction Log 2-32
Creating an Audit Trail 2-33
Recovering a Table 2-34
Strategies When Using Audit Trails 2-34
Locking 2-36
Table-level Locking 2-36
Row-level Locking 2-37
Wait for Locked Row 2-39
NULL Values 2-40
Default Values 2-40
The NULL in Expressions 2-41
The NULL in Boolean Expressions 2-42
The NULL in WHERE Clauses 2-43
The NULL in ORDER BY Clauses 2-44
The NULL in GROUP BY Clauses 2-44
The NULL Keyword in INSERT and UPDATE 'h1m'Statements 2-44
No NULL Databases 2-45
Views 2-46
Creating and Deleting Views 2-47
Querying through Views 2-48

Modifying through Views 2-48
Privileges with Views 2-50
Data Constraints Using Views 2-50
Outer Joins 2-52
Table Access by ROWID 2-53
TODAY and USER Functions 2-54
RDSQL Statements 2-56
The SELECT Statement 2-145
SELECT Clause 2-150
FROM Clause 2-153
WHERE Clause 2-155
GROUP BY Clause 2-171
HAVING Clause 2-173
ORDER BY Clause 2-175
INTO TEMP Clause 2-177
UNION Operator 2-178
Aggregate and Date Functions 2-180

Chapter 3. The FORMBUILD Transaction Form Generator

Overview 3-5
PERFORM Error Messages 3-5
Demonstration Database Sample Forms 3-5
Creating and Compiling a Custom Form 3-6
Using Menus to Create a Form 3-6
Using the Operating System to Create a Form 3-8
Structure of a Form Specification File 3-10
DATABASE Section 3-12
SCREEN Section 3-13
TABLES Section 3-19
ATTRIBUTES Section 3-21
Display Field Order 3-21
Table Order 3-22
Fields Linked to Database Columns 3-23
Display-Only Fields 3-25
Joining Columns 3-28
ATTRIBUTES Syntax 3-31
INSTRUCTIONS Section 3-63
COMPOSITES 3-64

DELIMITERS 3-66
MASTER OF 3-68
Control Blocks 3-71
Action Syntax 3-89
The sample Form Specification File 3-101
The CUSTOMER INFORMATION Screen 3-103
The ORDER INFORMATION Screen 3-103

Chapter 4. The PERFORM Screen Transaction Processor

Overview 4-5
Running PERFORM 4-6
Accessing PERFORM from the Main Menu 4-6
The PERFORM Screen 4-8
The Information Lines 4-9
The Screen Form 4-11
Status Lines 4-12
Running Operating System Commands from 'h1m'PERFORM 4-13
Entering Data 4-14
Data Types 4-14
Special Functions 4-16
Positioning the Cursor 4-16
Field Editing 4-17
Display Field Order 4-19
Data Checking 4-20
User Access Privileges on Multi-User Systems 4-21
The Current List 4-23
Menu Options 4-24

Chapter 5. The ACE Report Writer

Overview 5-5
Creating and Compiling a Custom Report 5-5
Using the Menus to Create a Report 5-5
Creating a Report from the Command Line 5-8
Information About ACE 5-11
ACE Filename Conventions 5-11

Using Expressions in a Report Specification 5-12
ACE Error Messages 5-14
Demonstration Database Sample Reports 5-15
Structure of a Report Specification File 5-16
DATABASE Section 5-17
DEFINE Section 5-18
INPUT Section 5-24
OUTPUT Section 5-27
SELECT Section 5-37
FORMAT Section 5-41
Control Blocks 5-47
Statements 5-65

Chapter 6. User-Menu

Overview 6-5
Accessing a Menu 6-6
Using a Menu Within INFORMIX-SQL 6-6
Designing a Menu 6-8
Creating a Menu 6-10
Accessing PERFORM with the menuform Form 6-10
Entering Menu Data 6-12
Steps for Entering Your Own Data 6-16
Modifying a Menu 6-18
Menu Display Fields 6-18
Creating a Script Menu 6-29

Appendix A. The stores Database

Appendix B. Environment Variables

Appendix C. INFORMIX-SQL Reserved Words

Appendix D. The dbschema Utility

Appendix E. Examples

Appendix F. ASCII Chart

Appendix G. The bcheck Utility

Appendix H. The sqlconv Utility

Appendix I. The dbupdate Utility

Appendix J. The dbload Utility

Appendix K. System Catalogs

Appendix L. Complex Outer Joins

Appendix M. Accessing Programs from the Operating System

INFORMIX-TURBO and INFORMIX-MS-NET Error Messages

INFORMIX-SQL Error Messages

Index

Introduction

Introduction Table of Contents

What Is INFORMIX-SQL? 5

About This Manual 6

Conventions 8
- Syntax 8
- Identifiers 10
- Database, Form, Report, and Command Filenames 10
- File Extensions 11
- System Catalogs 12

The Demonstration Database 13
- Tables in the Database 13
- Sample Screen Forms 16
- Sample Reports 16
- Sample RDSQL Command Files 17

Terminal and Operating System Interactions 18
- Number of Open Tables 18
- Terminal Types (UNIX Systems Only) 18
- Text Editors 19
- Environment Variables 19
- User Access Privileges on Multi-user Systems 20

What Is INFORMIX-SQL?

INFORMIX-SQL is a database management system. In its simplest form, a database management system is a computer-based record-keeping system. It consists of useful programs or modules designed to perform data management tasks. A good database management system can substantially reduce the amount of time required to organize, store, and retrieve your information. It can summarize, group, and format your information in ways that would otherwise be tedious and prone to error.

With **INFORMIX-SQL** you can perform these tasks:

- Create databases and tables
- Design screen forms
- Enter and modify data using screen forms
- Load data from operating system files
- Run queries from screen forms or with an interactive query language
- Produce custom-formatted reports
- Design a custom menu that can include **INFORMIX-SQL** elements, system utilities, and other programs

About This Manual

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

This manual is divided into an introductory section, 6 chapters, 14 appendixes, a section containing error messages, and an index. The following list summarizes the contents of the *INFORMIX-SQL Reference Manual*:

- The Introduction introduces **INFORMIX-SQL**, discusses conventions used in this manual, describes the demonstration database, and tells you where to look for technical information about **INFORMIX-SQL**.

- Chapter 1, "The INFORMIX-SQL Main Menu," explains how to use the INFORMIX-SQL Main Menu and describes what each option on the menu does.

- Chapter 2, "The RDSQL Query Language," describes the **RDSQL** query-language statements and gives examples of how to use them.

- Chapter 3, "The FORMBUILD Transaction Form Generator," focuses on **FORMBUILD**, supplying the information needed to build a screen form.

- Chapter 4, "The PERFORM Screen Transaction Processor," considers each **PERFORM** menu option in detail, explaining how to enter, modify, remove, and retrieve data.

- Chapter 5, "The ACE Report Writer," lists the formatting features you can use with **ACE** to prepare custom reports.

- Chapter 6, "User-menu," describes the **User-menu** option and provides the information needed to build a menu.

- Appendix A contains the **stores** demonstration database used in the *INFORMIX-SQL Reference Manual.*

- Appendix B describes the environment variables used by **INFORMIX-SQL.**

- Appendix C lists **INFORMIX-SQL** reserved words.

- Appendix D describes the **dbschema** utility that produces the **.sql** script necessary to create all or various parts of a database.

- Appendix E contains the example forms and reports used in the *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual.*

- Appendix F is an ASCII chart.

- Appendix G describes the **bcheck** utility used to check and restore the integrity of your index files.

- Appendix H explains how to use the **sqlconv** utility to convert **INFORMIX** databases to Version 2.1 **INFORMIX-SQL** databases.

- Appendix I describes the **dbupdate** utility that updates Version 1.0 **INFORMIX-SQL** databases and makes them compatible with Version 2.1 **INFORMIX-SQL** databases.

- Appendix J describes the **dbload** utility used to load data from other database systems or from raw data files into **INFORMIX-SQL** databases.

- Appendix K lists the structure of the system catalogs.

- Appendix L discusses complex outer joins.

- Appendix M demonstrates how to access each **INFORMIX-SQL** program from the command line.

- The section on Error Messages contains a list of all error codes, explains their meaning, and suggests remedies for avoiding the errors.

- The Index includes both the *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual.*

Because this is a reference manual, instructions and explanations assume some knowledge of computers and database management systems. The *INFORMIX-SQL User Guide* offers introductory instructions about how to use **INFORMIX-SQL.**

Conventions

The rules governing statement syntax and the naming of identifiers, databases, forms, reports, and command files are described in the next section. The filename extensions and the system catalogs used by **INFORMIX-SQL** are also noted.

Syntax

The notational conventions used in syntax statements that appear in both the *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual* are discussed in this section.

ABC — Any **INFORMIX-SQL** term in a syntax statement printed in uppercase letters is a keyword. Enter it exactly as shown, disregarding case. For example,

CREATE TABLE *tablename*

means you must enter the keywords CREATE TABLE or create table without adding or deleting spaces or letters.

abc — Substitute a value for any term that appears in lowercase italic letters. In the previous example,

you should substitute a value for *tablename*. In each statement description, a section labeled "Explanation" describes the values you can substitute for italicized terms.

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

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

```
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.

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

```
{|ONE | TWO | THREE|}
```

means that you must enter ONE or TWO or THREE and that you may not enter more than one option.

<u>ABC</u> When one of several options is the default option, it appears underlined. For example,

```
[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 a series of statements can follow the one that is listed.

Identifiers

Identifiers include table and column names, among others. Each identifier can be up to 18 characters in length, and the first character of an identifier must be a letter. You can use letters, characters, and underscores (_) for the rest of the identifier.

Each column name in a table must uniquely identify a column, but there can be duplicate column names in a database. When you are using several tables in a query, form, or report, and the same column name appears in more than one table, preface the column name with the table name and a period (**table-name.column-name**) to specify the correct column.

Database, Form, Report, and Command Filenames

Names for databases, form specification files, report files, and RDSQL command files can be up to 8 (on DOS systems) or 10 (on UNIX systems) characters long. The first character of a name must be a letter. You can use letters, characters, and underscores (_) for the rest of the name.

File Extensions

INFORMIX-SQL creates an operating system directory for each database. The directory contains files for the tables and indexes in that database. The directory and files are labeled with three-letter extensions. The list below shows the file extensions for a database directory and its files.

database.dbs	The name of the database directory
table_id.dat	The data for a table
table_id.idx	The index information for a table

Forms, reports, and command files associated with the database are stored in the current directory:

formfile.per Contains the specifications for a screen form. You create this file by using INFORMIX-SQL or a system editor from the command line.

You must compile this file before you can use the screen form.

formfile.frm Contains the compiled form specification. INFORMIX-SQL uses this file; you never work directly with it.

report.ace Contains the specifications for a report. You create this file by using INFORMIX-SQL or a system editor from the command line.

You must compile this file before you can run the report.

report.arc Contains the compiled report specifications. INFORMIX-SQL uses this file; you never work directly with it.

command.sql Contains one or more RDSQL statements. You create this file by using INFORMIX-SQL or a system editor from the command line.

System Catalogs

In addition to the files holding table data and index information, each database directory contains 18 system catalog files. The catalogs keep track of the tables, columns, indexes, views, synonyms, and permissions in each database.

UNIX System Catalogs:

syscolauth.dat	syscolauth.idx
syscolumns.dat	syscolumns.idx
sysdepend.dat	sysdepend.idx
sysindexes.dat	sysindexes.idx
syssynonym.dat	syssynonym.idx
systabauth.dat	systabauth.idx
systables.dat	systables.idx
sysusers.dat	sysusers.idx
sysviews.dat	sysviews.idx

DOS System Catalogs:

SYSCOLAU.DAT	SYSCOLAU.IDX
SYSCOLUM.DAT	SYSCOLUM.IDX
SYSDEPEN.DAT	SYSDEPEN.IDX
SYSINDEX.DAT	SYSINDEX.IDX
SYSSYNON.DAT	SYSSYNON.IDX
SYSTABAU.DAT	SYSTABAU.IDX
SYSTABLE.DAT	SYSTABLE.IDX
SYSUSERS.DAT	SYSUSERS.IDX
SYSVIEWS.DAT	SYSVIEWS.IDX

Appendix K lists the structure of the system catalogs.

The Demonstration Database

The examples in the *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual* are based on the **stores** demonstration database. You create the **stores** database and its accompanying forms, reports, and command files in the current directory by entering isqldemo from the operating system command line. **INFORMIX-SQL** removes any existing database labeled **stores** and installs the demonstration package in your current directory.

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."

Tables in the Database

The **stores** database contains information about a fictitious wholesale sporting goods distributor that services stores in the western United States. The database is made up of five tables:

customer
orders
items
stock
manufact

The **customer** table contains information about 18 stores that order sporting goods from the distributor. This information includes the store number, name, address, and telephone number and the name of the store representative.

The **orders** table contains 15 orders placed by the customers in the **customer** table. This information includes the order number, date the order was made, the customer number, shipping instructions, whether a backlog exists, the customer purchase order number, date the order was shipped, weight of the order, shipment charge, and the date the customer paid for the order.

The **items** table keeps track of individual item lines in an order. For example, some orders only contain one item line. Other orders contain as many as four item lines. Information contained in the **items** table includes the item line number, the order number, the stock number of the item, the manufacturer's code, the quantity ordered, and the total price of the ordered items.

The **stock** table is a catalog of the articles sold by the distributor. Information in the table includes a stock number representing the type of sporting item (for example, a football), a code identifying the manufacturer, a description of the item, its unit price, the unit by which the item must be ordered (by individual piece or by the case), and a description of the unit (for example, a case containing ten baseballs).

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 structure and content of the **stores** database are displayed in Appendix A. The potential relationships across the columns in the five tables are visually portrayed in the figure that follows.

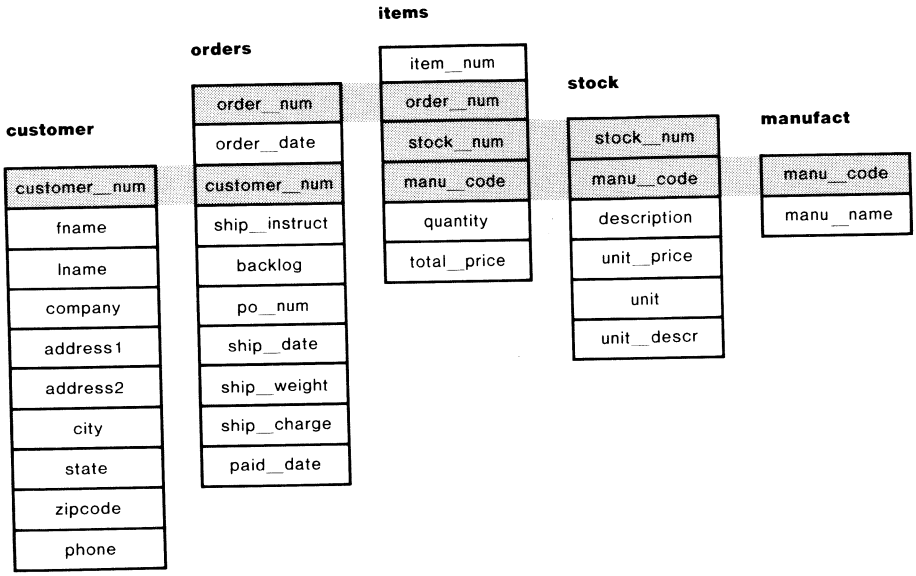


Figure Introduction-1. Join Relationships in the **stores** Database (potential join columns shaded)

Sample Screen Forms

The examples in this manual are based on the **customer, orderform,** and **sample** forms. Other forms have been included with the demonstration database to provide further practice once you have become familiar with these examples. These forms illustrate a variety of screen form features available in INFORMIX-SQL.

customer	Demonstrates the use of basic FORMBUILD attributes. It uses data in the **customer** table.
orderform	Demonstrates a multiple-table form. It uses data in the **customer**, **orders**, **items**, and **manufact** tables.
sample	Demonstrates several of the advanced features of PERFORM. It uses data in all five tables.

Sample Reports

The examples in this manual, which illustrate a variety of report features, are based on the reports listed below. Other reports have been included with the demonstration database to provide further study after you have become familiar with these samples.

mail1	A simple report that generates mailing labels
mail2	A more sophisticated report that produces one column of mailing labels
mail3	An interactive report that produces one to three columns of mailing labels
clist1	A report that lists customer information
clist2	An interactive customer report

ord1	A report that lists customer orders
ord2	A second customer order report
ord3	An interactive report that lists daily orders

Sample RDSQL Command Files

The demonstration database includes several **RDSQL** command files. The **RDSQL** commands used to create the **stores** database, the five tables, and the indexes on the tables are in these files.

c_stores	Creates the **stores** database
c_custom	Creates the **customer** table
c_items	Creates the **items** table
c_manuf	Creates the **manufact** table
c_orders	Creates the **orders** table
c_stock	Creates the **stock** table
c_index	Creates the indexes on the **customer**, **items**, **manufact**, **orders**, and **stock** tables

Nineteen files named **ex1** through **ex19** are also included. These files contain scripts that correspond to the examples found in Chapter 7 of the *INFORMIX-SQL User Guide.* Additional files are included for further exploration once you have become familiar with these samples.

Terminal and Operating System Interactions

The performance of INFORMIX-SQL depends in part upon the operating system and the specification of environment variables.

Number of Open Tables

The number of tables INFORMIX-SQL can open simultaneously determines the number of tables you can use in a cross-table query. This figure is machine dependent.

Terminal Types (UNIX Systems Only)

To run the screen-oriented programs used in INFORMIX-SQL you have to define the type of terminal you are using. Incorrect or non-existent terminal definitions may cause problems in displays. If the **vi** editor works correctly, your terminal definition is probably correct.

Terminal definitions are normally located in the **/etc/termcap** or **/usr/lib/terminfo** file. If you do not have a **termcap** or **terminfo** file on your system, or your terminal type is not included in the file, you can use the **termcap** file included with your INFORMIX-SQL software. See the "Setting Up the Operating Environment" section in the Preface to the *INFORMIX-SQL User Guide.*

Text Editors

You need a text editor to prepare and to modify

- **PERFORM** form specifications
- **ACE** report specifications
- **RDSQL** command files

RDSQL provides an internal text editor for your editing tasks: the **RDSQL** editor. The **RDSQL** editor handles as many lines of text as you want to enter; you are limited only by the memory constraints of your system. If you insert more lines than one screen can hold, the **RDSQL** editor scrolls down the page with the additional text.

If you prefer working with a system editor, you can select the RDSQL Menu **Use-editor** option.

INFORMIX-SQL automatically provides a default operating system editor. You can specify a different text editor with the DBEDIT environment variable. On UNIX systems, **vi**, **ex**, or **ed** are common text editors. On DOS systems, **edlin** or any word processing program that does not generate invisible control characters is appropriate. See Appendix B, "Environment Variables," for more information on how to specify a system editor.

Environment Variables

Environment variables affect the computing environment in which **INFORMIX-SQL** operates. Several of the more prominent environment variables used by **INFORMIX-SQL** are included in the following list. The complete set is described in Appendix B.

DBDELIMITER	specifies the field delimiter to use for LOAD and UNLOAD statements. The vertical bar (|) is the default.

DBEDIT	names the text editor to use when you create forms, reports, or command files. The default is **vi** for UNIX systems and **edlin** for DOS systems.
DBPATH	tells INFORMIX-SQL in what directories your databases are located. Make sure you set the DBPATH environment variable before you execute the **startsql** command.
DBPRINT	names the print command for your computer. The default is **lp** (for UNIX systems) or **lpt1** (for DOS systems).
DBTEMP	specifies the location of temporary files generated by INFORMIX-SQL. If you do not specify a location, **/tmp** is the default on UNIX systems. Your current directory is the default on DOS systems.
INFORMIXDIR	specifies the location of program files in INFORMIX-SQL. These files are usually stored in subdirectories of **/usr/informix** (UNIX systems) or **\informix** (DOS systems). You must set INFORMIXDIR only if the files are stored elsewhere.

User Access Privileges on Multi-User Systems

You can specify user access privileges for a database and the individual tables it contains with the RDSQL GRANT and REVOKE statements. Until permission is granted to others, the person who creates a database is the only one who can access the database. Users who have not been granted database access privileges cannot use the database. Access to individual tables can also be controlled. See Chapter 2, "The RDSQL Query Language," for details.

Chapter 1

The INFORMIX-SQL Main Menu

Chapter 1 Table of Contents

Accessing INFORMIX-SQL on DOS Systems 5
  Loading the Database Agent 5
  Entering INFORMIX-SQL 6

Accessing INFORMIX-SQL on UNIX Systems 6

The INFORMIX-SQL Screens 6
  Menu Screens 7
    Selecting Options 7
    Exiting the Menu 7
    Asking for Help 7
  Text-Entry Screens 8
    Entering Text 8
    Exiting a Text-Entry Screen 8
    Asking for Help 9

Maps of the Menu Structure 9

The INFORMIX-SQL Main Menu Options 11
    DATABASE 12
    EXIT 14
    FORM 15
    QUERY LANGUAGE 17
    REPORT 19
    TABLE 21
    USER-MENU 23

Accessing INFORMIX-SQL on DOS Systems

Before working with INFORMIX-SQL, be sure your computer is running properly and the operating system prompt appears on the screen. To begin accessing INFORMIX-SQL, 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. When you do this, you free memory for running other programs on your system.

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

After you have loaded the database agent, enter

```
isql
```

If you try to run isql without 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

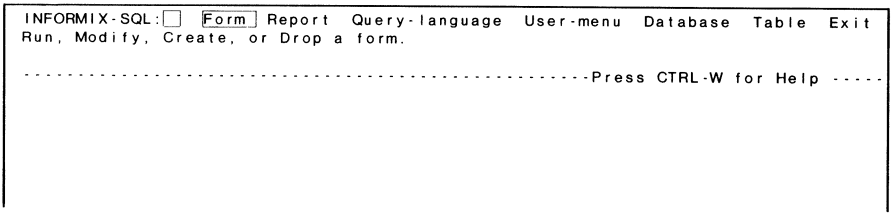
```
set
```

This gives you a list of your environment variables. If the database agent is 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
```

Accessing INFORMIX-SQL on UNIX Systems

To begin working with **INFORMIX-SQL**, enter isql at the operating system prompt. When you have successfully entered isql, **INFORMIX-SQL** displays the INFORMIX-SQL Main Menu:



The INFORMIX-SQL Screens

The **INFORMIX-SQL** menu system uses two kinds of screens: a menu screen like the INFORMIX-SQL Main Menu and a text-entry screen.

The top line of a menu screen lists your options. One option is always highlighted. The second line gives a brief description of the highlighted option. Each time you press the SPACEBAR, the highlight moves to the next option and the description changes. You can also use the RIGHT ARROW and LEFT ARROW keys to move the highlight. The fourth line displays the name of the current database and the message Press CTRL-W for Help. (On DOS systems, the message is Press F1 for Help.)

Selecting Options

You can normally select menu options in two ways:

- Use the SPACEBAR to move the highlight over the option you want to choose and press RETURN.

- Type the first letter of the option you want to select. Case is not important—you can type t or T to select the **Table** option.

INFORMIX-SQL displays the screen for the menu option you have selected.

Exiting the Menu

Each menu has an **Exit** option. When you want to leave a menu screen, type e for **Exit.** INFORMIX-SQL displays the previous menu or screen.

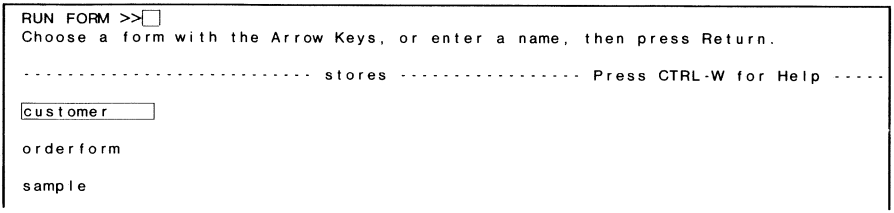
Asking for Help

The CTRL-W key (or F1 key on DOS systems) displays the help message appropriate for each part of INFORMIX-SQL. When you are finished reading the message displayed on the HELP Screen, press RETURN. INFORMIX-SQL redisplays the screen you were working with before calling for help.

Text-Entry Screens

The text-entry screen is the second kind of screen. It requires that you enter text instead of choosing a menu option. The top line of the screen displays the screen name, followed by double angle >> brackets. The second line gives directions. The fourth line displays the name of the current database and the message Press CTRL-W for Help. (On DOS systems, the message is Press F1 for Help.)

The RUN FORM Screen is an example of a text-entry screen. Some of the items it includes are shown below.



Entering Text

Whatever you type appears after the double angle brackets at the top of the screen. Press the **RETURN** key when you are finished typing. Some screens, like the RUN FORM Screen, give you the option of selecting an item from a list on the lower part of the screen instead of typing your selection. Use the **ARROW** keys to position the highlight over the item you want, and then press **RETURN**. **INFORMIX-SQL** displays the next screen.

Exiting a Text-Entry Screen

Text-entry screens do not have an **Exit** option. Press the program interrupt key (**DEL** on most UNIX systems or **CTRL-C** on DOS systems and BSD UNIX), and **INFORMIX-SQL** redisplays the previous menu or screen.

Asking for Help

The CTRL-W key (or F1 key on DOS systems) works with text-entry screens exactly as it does with menu screens. When you are finished reading the Help message, press RETURN. INFORMIX-SQL redisplays the screen you were working with before calling for help.

Maps of the Menu Structure

The INFORMIX-SQL Main Menu has seven options: **Form**, **Report**, **Query-language**, **User-menu**, **Database**, **Table**, and **Exit**. Each option on the Main Menu calls a submenu, which displays options that allow you to work with a part of INFORMIX-SQL. The INFORMIX-SQL menu structure is displayed in the following two figures.

Figure 1-1 is a map of the INFORMIX-SQL menu hierarchy. This figure illustrates the options on each of the submenus available from the Main Menu.

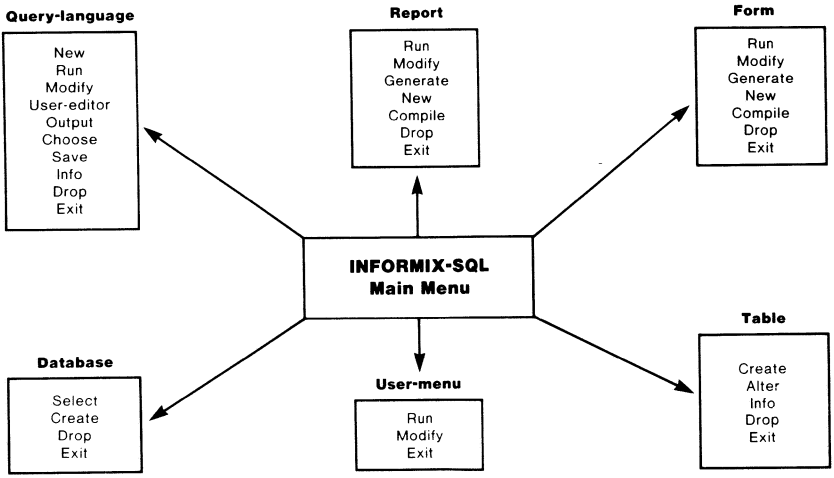


Figure 1-1. The INFORMIX-SQL Menu Hierarchy

Figure 1-2 is a functional guide to the INFORMIX-SQL menu system. Menu options are grouped according to activity or task.

INFORMIX-SQL Menu Structure—A Functional Map

Menu	Form	Report	RDSQL	User-menu	Database	Table
Function						
Use it:	Run	Run	Run	Run	Select	
Modify it:	Modify	Modify	Modify	Modify		Alter
Create it						
Default:	Generate	Generate	New		Create	
Custom:	New	New	Use-Editor			Create
Compile it:	Compile	Compile				
Special			Info			Info
Tasks:			Output			
			Save			
Drop it:	Drop	Drop	Drop		Drop	Drop
Exit:	Exit	Exit	Exit	Exit	Exit	Exit

Figure 1-2. A Functional Guide to the Menu Hierarchy

The INFORMIX-SQL Main Menu Options

The following sections discuss the options on the INFORMIX-SQL Main Menu. The options are presented in alphabetical order.

DATABASE

Overview

Use the **Database** option to create a new database, make an existing database the current database, or drop an existing database.

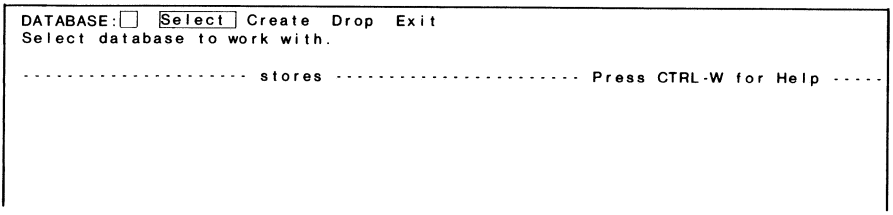


Figure 1-3. The DATABASE Menu

Procedure

The DATABASE Menu displays four options:

Select	Makes a database the current database
Create	Creates a new database and makes that database the current database
Drop	Removes a database from the system
Exit	Exits the DATABASE Menu and returns to the INFORMIX-SQL Main Menu

Notes

1. When you create a database through the **Create** option, that database becomes the current database.

2. When using the **Select** option, you must enter the name of a database located in the current directory or a directory specified in your DBPATH environment variable. If you enter the name of a non-existent database or a database that INFORMIX-SQL cannot locate, INFORMIX-SQL displays the message

   ```
   Database not found
   ```

3. Be very careful when you drop a database; all data in the database is permanently discarded.

4. Chapter 2, "Creating a Database," and Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide* describe the use of options on the DATABASE Menu.

5. Chapter 2, "The RDSQL Query Language," contains a list of all RDSQL database statements.

6. You are not allowed to drop the current database. You must close it first.

EXIT

Overview

Use the **Exit** option to leave the INFORMIX-SQL Main Menu and return to the operating system.

Procedure

1. From the INFORMIX-SQL Main Menu, type e to select the **Exit** option.

2. You leave the INFORMIX-SQL Main Menu and return to the operating system.

FORM

Overview

Use the **Form** option to run a screen form, create or modify a screen form, compile a screen form, or drop an existing screen form.

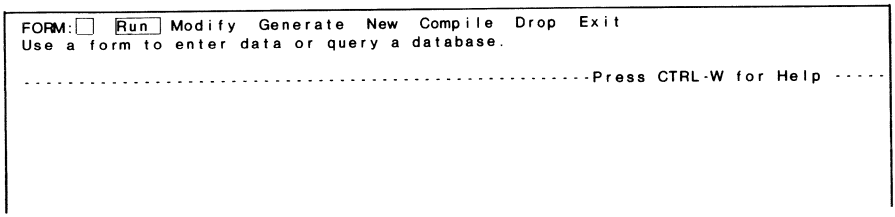


Figure 1-4. The FORM Menu

Procedure

The FORM Menu displays seven options:

Run	Runs a previously compiled screen form
Modify	Modifies a screen form specification
Generate	Creates a default screen form
New	Creates a custom screen form specification
Compile	Compiles a screen form specification
Drop	Drops a screen form
Exit	Exits the FORM Menu and returns to the INFORMIX-SQL Main Menu

Notes

1. After editing a form specification file (with the **New** or **Modify** options on the FORM Menu), you must compile it. (The form cannot be used in **INFORMIX-SQL** until it has been compiled.) Menus allowing you to compile an edited form are displayed when you select the **New** or **Modify** options. The **Compile** option on the FORM Menu can also be used to compile a form specification.

   INFORMIX-SQL notifies you if there are errors in the form specification. Follow the directions on the screen for correcting and recompiling the form. You can save or discard the form after compilation. You can also save an uncompiled form to work on at a later time or discard it completely.

2. Chapter 3, "Entering Data," Chapter 4, "Querying a Database," Chapter 5, "Using Multiple-Table Forms," and Chapter 6, "Creating Your Own Forms," in the *INFORMIX-SQL User Guide* describe how to create and use screen forms.

3. The **INFORMIX-SQL** program that compiles a form specification is called **FORMBUILD**. See Chapter 3, "The FORMBUILD Transaction Form Generator," for information about **FORMBUILD**.

4. The **INFORMIX-SQL** program that runs a screen form is called **PERFORM**. See Chapter 4, "The PERFORM Screen Transaction Processor," for information about **PERFORM**.

QUERY LANGUAGE

Overview

Select the **Query-language** option to use the **RDSQL** query language.

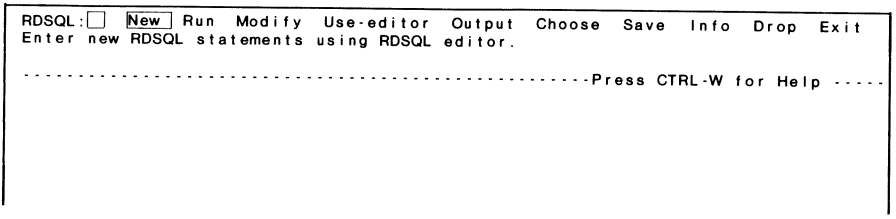


Figure 1-5. The RDSQL Menu

Procedure

The RDSQL Menu displays ten options:

New Allows you to enter new **RDSQL** statements using the **RDSQL** editor.

Run Executes the current **RDSQL** statement or statements.

Modify Allows you to use the **RDSQL** editor to modify the current **RDSQL** statement or statements.

Use-editor Allows you to enter or edit **RDSQL** statements with a system editor.

Output Routes the output from executing the current **RDSQL** statements to a system file, a printer, or (on UNIX systems only) a system pipe.

Choose Allows you to select an existing command file containing **RDSQL** statements and make them your current statements. The current statements can be run or edited.

Save Saves the current **RDSQL** statements in a command file. This command file can be used later by selecting the **Choose** option on the RDSQL Menu.

Info Allows you to retrieve information about the columns, indexes, privileges, and status of a table.

Drop Drops a command file from the database.

Exit Exits the RDSQL Menu and returns to the INFORMIX-SQL Main Menu.

Notes

1. If there is no current database, **INFORMIX-SQL** displays the CHOOSE DATABASE Screen after you select the **Query-language** option on the INFORMIX-SQL Main Menu.

2. Chapter 7, "Using RDSQL," in the *INFORMIX-SQL User Guide* describes how to use the RDSQL Menu and how to create and run **RDSQL** statements.

3. See Chapter 2, "The RDSQL Query Language," for complete information about **RDSQL**.

REPORT

Overview

Use the **Report** option to run a report, create or modify a report, compile a report, or drop an existing report from the database.

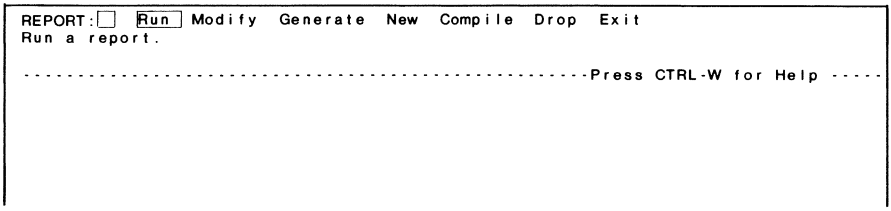


Figure 1-6. The REPORT Menu

Procedure

The REPORT Menu displays seven options:

Run	Runs a report
Modify	Modifies a report specification
Generate	Creates a default report specification
New	Creates a custom report specification
Compile	Compiles a report specification
Drop	Drops a report specification from the database
Exit	Exits the REPORT Menu and returns to the INFORMIX-SQL Main Menu

Notes

1. After editing a report specification file (with the **New** or **Modify** options on the REPORT Menu), you must compile it. (The report cannot be used in INFORMIX-SQL until it has been compiled.) Menus allowing you to compile an edited report are displayed when you select the **New** or **Modify** options. The **Compile** option on the REPORT Menu can also be used to compile a report specification.

   INFORMIX-SQL notifies you if there are errors in the report specification. Follow the directions on the screen for correcting and recompiling the report. You can save or discard the report after compilation. You can also save an uncompiled report to work on later or discard the report completely.

2. Chapter 8, "Creating and Printing Reports," in the *INFORMIX-SQL User Guide* describes how to create and use reports.

3. The INFORMIX-SQL program that compiles a report specification is called ACEPREP. The INFORMIX-SQL program that runs a report specification is called ACEGO. See Chapter 5, "The ACE Report Writer," for complete information about these programs.

TABLE

Overview

Use the **Table** option to create or modify a table, retrieve information about a table, or drop a table from the database.

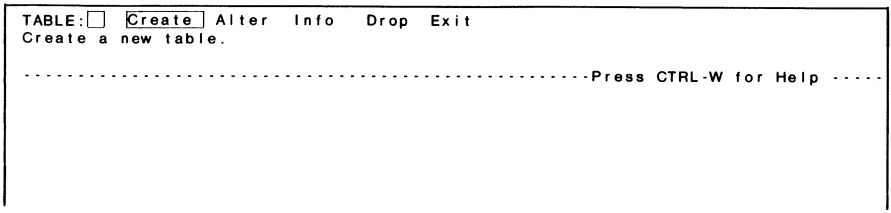


Figure 1-7. The TABLE Menu

Procedure

The TABLE Menu displays five options:

Create	Allows you to use the interactive schema editor to create a new table
Alter	Allows you to modify a table using the interactive schema editor
Info	Retrieves information about the structure of a table
Drop	Deletes a table from the database
Exit	Exits the TABLE Menu and returns to the INFORMIX-SQL Main Menu

Notes

1. If there is no current database, the CHOOSE DATABASE Screen appears after you select the **Table** option.

2. Be very careful when you drop a table; you lose all the data in the table.

3. Chapter 2, "Creating a Database," and Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide* describe the use of options on the TABLE Menu.

4. See Chapter 2, "The RDSQL Query Language," for information about the **RDSQL** table statements.

USER-MENU

Overview

Use the **User-menu** option to run a user-created menu, create a user-menu, or modify an existing user-menu.

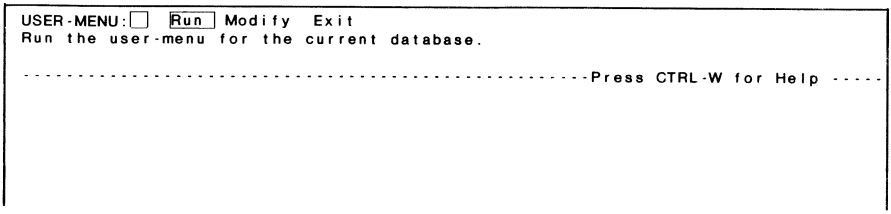


Figure 1-8. The USER-MENU Menu

Procedure

The USER-MENU Menu displays three options:

Run	Runs the user-menu for the current database
Modify	Allows you to create or modify a user-menu
Exit	Exits the USER-MENU and returns to the INFORMIX-SQL Main Menu

Notes

1. After you select the **User-menu** option, the CHOOSE DATABASE Screen appears if there is no current database.

2. Use the **Modify** option to both create and modify a user-menu.

3. If there is no user-menu for the current database, **INFORMIX-SQL** displays a message.

4. See Chapter 6, "User-menu," for complete information about creating, modifying, and using a menu.

Chapter 2

The RDSQL Query Language

Chapter 2 Table of Contents

Overview 7

How to Use RDSQL 8

Overview of RDSQL 10
- Formatting RDSQL Statements 10
- RDSQL Identifiers 11
- Database Data Types 11

RDSQL Statement Summary 14
- Data Definition Statements 14
- Data Manipulation Statements 16
- Data Access Statements 17
- Data Integrity Statements 18
- Auxiliary Statements 19

User Status and Privileges 21

Indexing Strategy 23

Auto-Indexing 26

Clustered Indexes 26

Transactions 28
- Transaction Log File Maintenance 30

Audit Trails 32
- Comparing Audit Trails and the Transaction Log 32
- Creating an Audit Trail 33
- Recovering a Table 34
- Strategies When Using Audit Trails 34

Locking 36
- Table-level Locking 36
- Row-level Locking 37
  - Locking Groups of Rows 37
  - Locking Individual Rows 38

Wait for Locked Row 39

NULL Values 40
Default Values 40
Default Values in RDSQL Version 1.10 41
The NULL in Expressions 41
The NULL in Boolean Expressions 42
The NULL in WHERE Clauses 43
The NULL in ORDER BY Clauses 44
The NULL in GROUP BY Clauses 44
The NULL Keyword in INSERT and UPDATE Statements 44
No NULL Databases 45

Views 46
Creating and Deleting Views 47
Querying through Views 48
Modifying through Views 48
Privileges with Views 50
Data Constraints Using Views 50

Outer Joins 52

Table Access by ROWID 53

TODAY and USER Functions 54

RDSQL Statements 56
ALTER INDEX 57
ALTER TABLE 60
BEGIN WORK 63
CHECK TABLE 65
CLOSE DATABASE 67
COMMIT WORK 68
CREATE AUDIT 69
CREATE DATABASE 71
CREATE INDEX 74
CREATE SYNONYM 77
CREATE TABLE 79
CREATE VIEW 85
DATABASE 88
DELETE 90
DROP AUDIT 92

DROP DATABASE 93
DROP INDEX 95
DROP SYNONYM 96
DROP TABLE 97
DROP VIEW 99
GRANT 101
INFO 106
INSERT 108
LOAD 112
LOCK TABLE 115
OUTPUT 117
RECOVER TABLE 119
RENAME COLUMN 121
RENAME TABLE 123
REPAIR TABLE 125
REVOKE 127
ROLLBACK WORK 130
ROLLFORWARD DATABASE 131
SELECT 132
SET LOCK MODE 133
START DATABASE 135
UNLOAD 137
UNLOCK TABLE 139
UPDATE 140
UPDATE STATISTICS 144

The SELECT Statement 145
SELECT Clause 150
FROM Clause 153
WHERE Clause 155
Comparison Condition 155
Join Conditions 164
Subqueries 167
GROUP BY Clause 171
HAVING Clause 173
ORDER BY Clause 175
INTO TEMP Clause 177
UNION Operator 178
Aggregate and Date Functions 180
Aggregate Functions 181
DATE() 183
DAY() 184

MDY() 185
MONTH() 186
WEEKDAY() 187
YEAR() 188

Overview

RDSQL is an extension of the Structured Query Language (SQL) developed by IBM for creating and querying databases. It meets the proposed ANSI standards for SQL implementations. RDS additions to the language permit you to load and unload database tables into system files, to change databases, and to change the names of tables and columns. Other extensions increase the functionality of standard SQL statements.

RDSQL is the interactive query language for INFORMIX-SQL. RDSQL is also the language you use to select the data you want in a report. See Chapter 5, "The ACE Report Writer," for a detailed description of ACE, the INFORMIX-SQL report-writing program.

The first two sections of this chapter describe how you can access RDSQL and give an overview of this facility. A summary of the RDSQL statements follows. The next sections discuss database privileges, indexing strategies, transactions, audit trails, locking, NULL values, views, outer joins, and table access by ROWID.

All RDSQL statements are defined in detail in the section that follows. The statements are organized alphabetically. Syntax displays use the conventions defined in the Introduction.

This chapter concludes with a section describing the SELECT statement.

How to Use RDSQL

You can run **RDSQL** statements one at a time, or you can run a sequence of **RDSQL** statements separated by semicolons.

---

statement;
statement;
.
.
.
statement

With **RDSQL**, the semicolon is a separator, and not a terminator, for statements. You do not need a semicolon if you are running only one statement.

To work with **RDSQL**, select the **Query-language** option on the INFORMIX-SQL Main Menu. If there is no current database, **INFORMIX-SQL** displays the CHOOSE DATABASE Screen. Select a database, and the RDSQL Menu is displayed:

```
RDSQL: [ New ] Run  Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit
Enter new RDSQL statements using RDSQL editor.

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

While you are working with **INFORMIX-SQL**, **RDSQL** remembers the last sequence of statements, if any, with which you worked. These statements are called the current statements.

The RDSQL Menu displays ten options:

New Allows you to enter a new set of current statements. The previous current statements will be lost when you select the **New** option if you have not saved them with the **Save** option.

Run	Runs the current statements.
Modify	Displays the RDSQL editor with the current statements. You can correct syntax errors, make changes, or add additional statements.
Use-editor	Allows you to use a system editor with the current statements in the editor buffer. The default editor is **vi** (on UNIX systems) or **edlin** (on DOS systems). You can change the default editor by setting the DBEDIT environment variable (see Appendix B). When you write the file to disk, do not give it a name; RDSQL gives it a temporary name until you choose to save it as a system file.
Output	Sends the results of running the current statements to the printer, a system file, or (on UNIX systems only) another program.
Choose	Allows you to select the contents of a command file as the current statements. RDSQL presents a list of all command files (files with the extension **.sql**) in the current directory. If there is no current database, the list will contain all command files located in the current directory and the directory containing the current database.
Save	Saves the current statements in a command file. RDSQL adds the **.sql** filename extension to the filename you select.
Info	Displays information about a table in the current database. (See the INFO statement later in this chapter for details.)
Drop	Drops an RDSQL command file from the database.
Exit	Returns to the INFORMIX-SQL Main Menu.

You can also run RDSQL statements directly from the operating system command line. Appendix M describes how to do this.

Overview of RDSQL

RDSQL statements are used to create and manipulate relational databases. A relational database consists of one or more tables that, in turn, are constructed of rows and columns. Each row contains a particular set of column values. Databases are created as subdirectories of the current directory. The name of the directory is the database name, with the extension **.dbs**.

The database subdirectory contains eighteen system catalogs. INFORMIX-SQL uses the system catalogs to keep track of the tables, columns, indexes, views, synonyms, and permissions in each database. The database subdirectory also contains the tables that constitute the database. Each of these tables is represented by data files and index files with the extensions **.dat** and **.idx**, respectively. The system catalogs are described in Appendix K.

Formatting RDSQL Statements

RDSQL is a free-form language that treats all contiguous white space (containing spaces, tabs and/or NEWLINEs) as a single space. The sequence of keywords and data is important, but the physical arrangement of the statements on the screen is not significant to RDSQL. You should format the statements so that they are clear to you and to others. Command files may include comments (enclosed within { } braces) that are ignored by RDSQL.

RDSQL Identifiers

An **RDSQL** identifier is the name of an object and can consist of letters, numbers, and underscores (_). The first character must be a letter. Unless otherwise noted, an identifier can have from one to eighteen characters.

Database A database name is an identifier that can have from 1 to 8 (DOS) or 10 (UNIX) characters.

Table A table name is an identifier that must be unique within the database.

Column A column name is an identifier that must be unique within a table; there can be duplicate column names within a database. When the same column name appears in more than one table used in a query, use the notation *table.column* to specify the intended column.

Database Data Types

You must assign a data type to every column in the database (see the CREATE TABLE statement in the last section of this chapter). A detailed discussion of data types is contained in Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide.*

A brief set of guidelines describing valid **RDSQL** data types follows.

CHAR(*n*) A character string of length *n* (where 1 <= *n* <= 32,767).

SMALLINT A whole number from −32,767 to +32,767, inclusive.

INTEGER A whole number from −2,147,483,647 to +2,147,483,647, inclusive.

DECIMAL[(*m*[,*n*])]

A decimal floating-point number with a total of *m* (<= 32) significant digits (the precision) and *n* (<= *m*) digits to the right of the decimal point (the scale). When you give values for both *m* and *n*, operations on the DECIMAL variable will use "fixed point" arithmetic. All numbers less than 0.5×10^{-n} in absolute value have the value zero. The largest absolute value of a variable of this type that can be stored without an error is $10^{m-n} - 10^{-n}$.

The second parameter is optional and, if missing, the variable is treated as a floating decimal. This means that DECIMAL(*m*) variables have a precision of *m* and a range in absolute value from 10^{-128} to 10^{126}. When there are no other formatting instructions, DECIMAL(*m*) variables are printed with two decimal places to the right of the decimal point (and an exponent, if necessary). If no parameters are designated, DECIMAL is treated as DECIMAL(16), a number with 16 significant digits and an implied floating decimal point.

SMALLFLOAT

A binary floating-point number corresponding to the single-precision, floating-point data type. The range of values for a SMALLFLOAT data type is the same as the range of values for the "float" data type in the C language on your machine. On small machines, **RDSQL** implements SMALLFLOAT as DECIMAL(8).

FLOAT

A binary floating-point number corresponding to the double-precision, floating-point data type. The range of values for a FLOAT data type is the

	same as the range of values for the "double" data type in the C language on your machine. On small machines, **RDSQL** implements FLOAT as DECIMAL(16).
MONEY[(*m*[,*n*])]	Like the DECIMAL data type, the MONEY data type can also take two parameters. The limitation on values for columns of type MONEY(*m*, *n*) is the same as for columns of type DECIMAL(*m*, *n*). The difference between them is that MONEY type variables are displayed with a dollar sign. The type MONEY(*m*) is defined as DECIMAL(*m*, 2) and, if no parameter is given, MONEY defaults to DECIMAL(16, 2). Regardless of the number of parameters, the data type MONEY is always treated as a fixed-point decimal number.
SERIAL[(*n*)]	A unique sequential integer assigned automatically by **RDSQL**. You may assign an initial value *n*. The default starting value is 1.
DATE	DATE values must be enclosed within double quotation marks and can be expressed either with the format **mm/dd/yy** or with **mm/dd/yyyy**. **mm** stands for the month (1 or 01 for January, 2 or 02, for February, and so on). **dd** stands for the day of the month (from 1 to the maximum for that month). **yy** and **yyyy** stand for the year. When you use only two digits for the year, **RDSQL** assumes that you mean a year beginning with the digits "19." **RDSQL** stores a DATE value as an integer whose value is the number of days since December 31, 1899.

RDSQL Statement Summary

Five different types of **RDSQL** statements are used with **INFORMIX-SQL**:

- Data definition
- Data manipulation
- Data access
- Data integrity
- Auxiliary

Data Definition Statements

Data definition statements include those that create and drop a database and its tables, views, and indexes; and statements that modify or rename tables and columns. Of this list, only the DATABASE statement is required before manipulating the data of an existing database.

CREATE DATABASE — Creates a database directory, sets up the system catalogs, and makes the new database the current database. There can be no more than one current database at any time.

DATABASE — Selects a database and makes it the current database. There can be no more than one current database at any time.

CLOSE DATABASE — Closes the current database files and leaves no database current. Only three **RDSQL** statements are permitted when there is no current database:

- CREATE DATABASE
- DATABASE
- DROP DATABASE

DROP DATABASE	Deletes all tables, indexes, system catalogs, and the database subdirectory.
CREATE TABLE	Creates a table and defines the columns and their data types.
ALTER TABLE	Adds and drops columns from a table and modifies data types of columns.
RENAME TABLE	Changes the name of a table.
DROP TABLE	Deletes all data and indexes for a table and erases its entry in the system catalogs.
CREATE VIEW	Defines a table selected from rows and columns of existing tables and views. As the data from the underlying tables change, so does the view built upon them. (See the section "Views," later in this chapter, for information about how to use views.)
DROP VIEW	Deletes the definition of the view from the system catalogs, along with any views defined in relation to the one that is dropped. The underlying tables are unaffected.
CREATE SYNONYM	Defines an alternative name for a table or a view. A synonym is effective only for the user who creates it and can be dropped only by its creator. This means that if several people want to use the same synonym, they must each create the synonym.
DROP SYNONYM	Deletes a synonym from the system catalogs.

RENAME COLUMN	Changes the name of a column.
CREATE INDEX	Creates an index on one or more columns of a table. (See the section "Indexing Strategy," later in this chapter, for more information on indexes.)
ALTER INDEX	Allows you to modify a table so that the rows are in the same order as the index. The index is called a *clustered* index.
DROP INDEX	Deletes an index.
UPDATE STATISTICS	Updates statistics on tables in the system catalogs.

Data Manipulation Statements

The most frequently used statements are the data manipulation statements. These are as follows:

DELETE	Deletes one or more rows from a table
INSERT	Adds one or more rows to a table
SELECT	Retrieves data from a database
UPDATE	Modifies the data in one or more rows of a table

SELECT is the most important and the most complex **RDSQL** statement. Its syntax is defined at the end of this chapter. The following example illustrates its use.

```
SELECT lname, company
   FROM customer
   WHERE customer_num = 101;
```

This statement queries the **customer** table and returns the single row for which the customer number is 101. From that row, it selects and displays the values in the columns corresponding to the last name and company name.

Data Access Statements

A user has access to the database and to specific columns within a table only when the Database Administrator (DBA) or the owner of the table specifically grants these privileges. You can temporarily limit access to a table by executing the LOCK TABLE statement. (Under transactions, RDSQL locks only the affected rows until the transaction is complete. Explicit table/record locking is generally not required.)

RDSQL provides the following statements affecting data access:

GRANT	Grants database access privileges to specific users or to the public.
REVOKE	Removes database access privileges from specific users or from the public.
LOCK TABLE	Limits access to the table to the current user only or allows other users only to read the table. Use the LOCK TABLE statement only when making major changes to a table in a multi-user environment and when simultaneous interaction with the table by another user would interfere. LOCK TABLE decreases the accessibility of the database because it prevents other users from accessing the table. If the database has transactions, you must issue the BEGIN WORK statement before issuing a LOCK TABLE statement.
UNLOCK TABLE	Restores access to a previously locked table.

SET LOCK MODE Determines whether subsequent **RDSQL** calls wait for a locked row to become unlocked.

See the section "User Status and Privileges," later in this chapter, for further information.

Note: Data access statements are effective only on multi-user systems.

Data Integrity Statements

RDSQL provides data integrity by implementing the concepts of audit trails and transactions. An audit trail is a file that contains a history of additions, deletions, updates, and manipulations to a database table. In the event of a system failure, you can use the audit trail facilities to restore a table from a backup copy of the table and an audit trail file.

A transaction is a series of operations (**RDSQL** statements) on a database that you want completed as a single unit of work or not at all. For example, transactions are a necessity in the field of bookkeeping, where multiple operations on several different accounts must be made as a unit or the books will be out of balance. Using **INFORMIX-SQL**, you can achieve data integrity by maintaining a record of all operations that modify the contents of a database. You have the option to commit these changes to the database or to reverse the actions and leave the database untouched.

Both audit trails and transactions affect system performance and storage. For these reasons you should give careful consideration before using either. This topic is described in greater detail in the "Strategies When Using Audit Trails" section later in this chapter.

The following statements are used in the process of maintaining data integrity:

CREATE AUDIT Creates an audit trail file

DROP AUDIT	Drops an audit trail file from the database
RECOVER TABLE	Restores a database table from a backup copy and an audit trail file
BEGIN WORK	Marks the beginning of a transaction
COMMIT WORK	Marks the end of a transaction by authorizing all changes to the database since the last BEGIN WORK statement. COMMIT WORK releases all row and table locks
ROLLBACK WORK	Marks the end of a transaction by revoking all changes to the database since the last BEGIN WORK statement
START DATABASE	Initiates a new transaction log file
ROLLFORWARD DATABASE	Uses a transaction log file to restore a database from the last backup copy

See the "Audit Trails" and "Transactions" sections later in this chapter for more information.

Auxiliary Statements

The following statements are RDS extensions to SQL:

CHECK TABLE	Checks the indexes of a table to determine if they have become corrupted. This statement allows you to run the **bcheck** utility (Appendix G) from within **RDSQL**.
REPAIR TABLE	Repairs the indexes of a table if they have become corrupted. This statement allows you to run the **bcheck** utility (Appendix G) from within **RDSQL**.

LOAD	Reads an ASCII formatted system file into a database table. (See also the **dbload** utility, described in Appendix J.)
UNLOAD	Writes the contents of a database table in ASCII text format to a system file.
INFO	Displays database information held in the system catalogs. (See also the system catalog description in Appendix K.)
OUTPUT	Directs the output of a SELECT statement to a system file, a printer, or (on UNIX systems only) to a pipe.

User Status and Privileges

When you create a database, you are automatically the Database Administrator (DBA) of that database. At this point you are the only one who has access to the database. Another user cannot have access to a database until you grant the CONNECT privilege to that person. Another user cannot create or drop tables and indexes unless granted the RESOURCE privilege. Only the DBA can grant these privileges. You may also grant the DBA privilege to another user. The DBA privilege extends all the powers of the Database Administrator to the user, including the ability to alter the system tables; to drop, start, and roll forward the database; and to grant and revoke CONNECT, RESOURCE, and DBA privileges.

A user with the RESOURCE privilege will have the CONNECT privilege by default. A user with the DBA privilege will have both RESOURCE and CONNECT privileges. A DBA privilege can only be revoked by another DBA. A DBA cannot revoke the DBA privilege from himself or herself. If the creator of the database grants DBA privileges to another user, that user may revoke the DBA privilege from the database creator. This last property permits the transfer of authority from the maker of a database application to the person who will have the responsibility for maintaining the database.

An RDS extension to SQL allows the CONNECT and RESOURCE privileges to be granted to the PUBLIC, rather than to specifically named users.

In addition to these database-level privileges, there are several table-level privileges that can be granted by the owner of a table. These privileges permit a user to access specific columns for the purpose of executing SELECT or UPDATE statements and allow a user to insert new rows, delete old rows, create indexes, and alter the structure of the table.

Several of the RDSQL statements may be executed only by the DBA or by the owner of the table or index specified in the statement. These statements are:

ALTER TABLE
ALTER INDEX
DROP TABLE
DROP INDEX
DROP VIEW
RENAME COLUMN
RENAME TABLE
GRANT
REVOKE

(You can give others the privilege of executing the ALTER TABLE, GRANT, and REVOKE statements with certain restrictions.) The owner of a table is the person who executed the CREATE TABLE statement. The owner of an index is the one who executed the CREATE INDEX statement.

Note: In the INFORMIX-SQL demonstration database, RESOURCE privileges have been granted to PUBLIC.

Indexing Strategy

There are two major purposes for creating an index on columns of a database table:

- To speed sorting of rows
- To optimize the performance of queries

When your INFORMIX-SQL activities involve complex queries through a large database, significant time savings can result from judicious indexing. The drawback to having indexes is that they slow down the process of inserting, modifying, or deleting data. When you update a table, its indexes must also be modified. This is not normally a problem when you are adding information interactively, a row at a time, but can be time-consuming when you are inserting a large numbers of rows from one table into another.

The solution to this potential conflict between needs is to take a dynamic approach to indexing. One of the advantages of a relational database built upon C-ISAM is that issues like indexing can be handled with a great deal of flexibility. You need not decide upon indexes when the database is created, but can write your applications to create indexes when you need them and drop them when they get in the way. By judicious scheduling, you can create your indexes in anticipation of batch report writing (during, say, the night) and drop them the next morning (before there are huge data-entry needs).

Since it takes time to create an index on a table already containing data, you should create only those indexes that will optimize the queries you will be making. The following are hints for strategic indexing.

- Do not create indexes for tables with fewer than 200 rows. The speed you gain from using an index does not overcome the time required to open and search the index file on small tables.

- Do not create indexes on a column that has only a few possible values. Such columns are those that contain data like sex, marital status, yes/no responses, or zip codes in a small city. Because data like this produces skewed indexes, indexing can defeat the optimizing strategy of RDSQL. Queries may take longer than if the columns were unindexed. If you have a frequent need to have data sorted on columns with a small range of possible values, create a cluster index on the column.

- If the WHERE clause of a SELECT statement imposes a condition on a single column, put an index on that column. If there are conditions placed on several columns, make a composite index on all the affected columns. For the SELECT statement

```
SELECT * FROM items WHERE order_num > 1015;
```

  put an index on the **order_num** column.

  For the statement

```
SELECT * from items
   WHERE total_price > 1000.00 AND
      manu_code matches "HRO";
```

  create a composite index on both the **total_price** and **manu_code** columns.

- If the WHERE clause of a SELECT statement has a join condition between a single column in one table and a single column in another table, create an index on the column in the table with the larger number of rows. If several columns of one table have join conditions with several columns in another table, create a composite index on the affected columns of the table with the larger number of rows. For the SELECT statement

```
SELECT * FROM items, stock
   WHERE items.stock_num = stock.stock_num;
```

  place an index on the **stock_num** column in the **items** table, since it has many more rows than the **stock** table.

For the statement

```
SELECT * FROM items, stock
    WHERE items.stock_num = stock.stock_num
    AND items.manu_code = stock.manu_code;
```

put a composite index on the **stock_num** and **manu_code** columns in the **items** table.

- The speed with which the **dbload** utility loads data into a database is greatly affected by the presence of indexes. You may want to drop any indexes before running **dbload**, then replace them once the program is finished.

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. This operation is transparent except for the resulting dramatic improvement in the speed of unindexed queries.

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 will proceed. You can, at least temporarily, cause the physical ordering in the table to be the same as the ordering in an index through *clustering.*

RDSQL orders, or clusters, the physical data in a table when you create a new index by executing a variant of the CREATE INDEX statement or when you execute the ALTER INDEX statement (explained later in this chapter) for an existing index. Since users who have access to the table can add additional rows or update the information in existing rows, a table that you cluster according to an index will not stay that way. Over time, you can expect the benefit of an earlier clustering to disappear and you may want to cluster the table again using an ALTER INDEX TO CLUSTER statement.

Since a table can have only one physical ordering, you can have only one clustered index on a table at any given time. You can change the physical ordering to reflect a different index by executing two ALTER INDEX statements:

1. First execute an ALTER INDEX TO NOT CLUSTER statement to release the cluster attribute from the first index.

2. Next execute an ALTER INDEX TO CLUSTER statement to attach the cluster attribute to the second index.

You may not execute the ALTER INDEX or CREATE INDEX statements on a view.

Transactions

A transaction is a series of operations (RDSQL statements) on a database that you want to be completed entirely or not at all. RDSQL provides several statements that give you the ability to protect the integrity of your database by treating a transaction as a single unit of work to be processed.

The first statement is the CREATE DATABASE statement, which includes the WITH LOG IN clause. This statement creates not only the database, but also a transaction log file that keeps track of all modifications to the database.

When you want to perform a series of operations that you consider a unit, you issue the BEGIN WORK statement. This statement causes all subsequently altered rows of the database tables to be locked against modification by other users (although others may view them). When you are satisfied that the series of operations has produced the desired results, you terminate the transaction with a COMMIT WORK statement. If you are not satisfied with the results, you can terminate the transaction with a ROLLBACK WORK statement.

This statement restores the database to the state that existed when you issued the BEGIN WORK statement, *with an important exception.* All database definition statements that alter the number and names of tables, the number and names of columns, data types, and indexes are treated as singleton transactions. If they were executed successfully, they are committed and are not rolled back.

Both the COMMIT WORK and the ROLLBACK WORK statements unlock the rows and make them accessible for modification by others.

The number of rows that can be locked at one time by all users is limited. The actual limit depends on your operating system. You will be prudent if you restrict the definition of a transaction to a few statements that involve only a few rows. If you

expect that the number of rows entering into the transaction will be large, LOCK the table(s) involved until the transaction is completed. The LOCK TABLE statement must be issued after the BEGIN WORK statement.

The transaction log file can quickly become very large, and you may want to archive it on tape and initiate another log file. When you create a new log file, you must also create a backup copy of your database. Generally speaking, every log file must have a corresponding archive copy of the database. If your database was originally created without a transaction log or if you want to rename the transaction log file, you may create a new transaction log file for it by executing the START DATABASE statement and naming a new full filename for the log file.

If you have made a backup copy of your database and have a transaction log file that begins with the operations executed immediately after the backup was made, the DBA can bring the backup database up to date with the ROLLFORWARD DATABASE statement. This statement will recover the database through the last committed transaction.

If you intend to change the name of your log file, you must execute the START DATABASE statement just before you make your backup of the database. After rolling the database forward, the DBA is the only one who has access to the database, since it is left in an exclusive mode. This state allows the DBA to check the database for errors before making it generally available. The DBA must close the database when satisfied that the database has been restored correctly.

Note: The BEGIN WORK, COMMIT WORK, and ROLLBACK WORK statements cannot work unless you created the database with the WITH LOG IN option or you have subsequently executed the START DATABASE statement and named a transaction log file. A database created without a transaction log is described as a database "without transactions." A database created with a transaction log is described as a database "with transactions."

If you create a database with transactions, but do not execute the BEGIN WORK statement, RDSQL treats each statement as a singleton transaction. Each statement, if it executes successfully, is committed, and the database is permanently altered. If the statement fails, there is an automatic rollback to the status before the statement.

You cannot ROLLBACK any of the data definition statements, nor the GRANT or REVOKE statements. Do not use these statements within a transaction.

If you create a database without transactions, there is no automatic recovery from, say, an UPDATE statement that fails before completing its process.

Transaction Log File Maintenance

The transaction log file can become quite large and, periodically, the DBA may want to archive it on tape and initiate another log file. At the same time, the DBA should also create a backup of your database. Generally speaking, every log file must have a corresponding archive copy of the database.

The DBA should perform the following sequence of steps to back up a database:

1. Execute a DATABASE EXCLUSIVE statement to prevent other users from modifying the database.

2. Copy the transaction log file and the database directory onto a backup medium.

3. Create an empty log file with the same name as the old file.

4. Execute a CLOSE DATABASE statement.

The DBA can use the ROLLFORWARD DATABASE statement to bring the backup database up to date. The following two items must exist:

- A backup copy of the database
- A transaction log file that begins with the operations executed immediately after the backup was made

The DBA must load the backup database files and execute the ROLLFORWARD DATABASE statement.

The ROLLFORWARD DATABASE recovers the database through the last committed transaction. After rolling the database forward, the DBA is the only one who has access to the database, since it is left in an exclusive mode. This state allows the DBA to check the database for errors before making it generally available. The DBA must close the database once satisfied that the database has been restored correctly.

The DBA must perform the following sequence of steps if the database is without transactions and you want to begin using transactions, or if you want to change the name of the log file.

1. Execute a START DATABASE WITH LOG IN statement. Assign a new filename to the log file if you are changing its name.
2. Execute a CLOSE DATABASE statement.
3. Execute a DATABASE EXCLUSIVE statement.
4. Copy the database directory onto a backup medium.
5. Execute a CLOSE DATABASE statement.

Note: It is extremely important that the DBA execute the START DATABASE statement immediately before backing up the database. Otherwise, the ROLLFORWARD DATABASE statement cannot work.

Audit Trails

An audit trail is a file that contains a history of all additions, deletions, updates, and manipulations to a database table. It 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.

INFORMIX-SQL provides three audit trail statements to protect the integrity of a table:

CREATE AUDIT
DROP AUDIT
RECOVER AUDIT

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," and the transaction log cannot be dropped without extensive reworking of the database. You may add and delete audit trails as needed.

- To make full use of transactions, you must first issue a BEGIN WORK statement, perform a series of database modifications, and finish with a COMMIT WORK or ROLLBACK WORK statement. An audit trail automatically records table modifications.

- If you do not issue a BEGIN WORK and COMMIT WORK/ROLLBACK WORK set of statements, the transaction log records the results of each RDSQL statement as a singleton transaction. An audit trail acts in a similar manner, recording each modification to a table.

Creating an Audit Trail

Use the CREATE AUDIT statement to create an audit trail file and to begin writing the audit trail. The format is

CREATE AUDIT FOR *table-name* IN "*path-name*"

where *table-name* is the name of the table for which you want to create an audit trail file, and *path-name* is the full pathname of the audit trail file. The audit trail file should be on a physical device other than the one that holds the data, so that a system failure affecting the device that holds the data does not also damage the audit trail. If your computer system has more than one hard disk, you should write the audit trails to one of the disks not containing the data.

To use the audit trail, make a backup copy of the table *after* you have executed the CREATE AUDIT statement but *before* you have made any changes to the table. Once you have started the audit trail and have made a backup, you are ready to work with the table.

You can drop and create an audit trail file whenever you want. Drop and create the audit trail files whenever you make a complete backup of the device containing the data file. If a system failure should occur, you can use the audit trail to recover the table from the time of the last backup to the time the failure occurred.

Recovering a Table

In the event of a system failure, you can use the RECOVER TABLE statement to restore a database table from a backup copy and an audit trail file that was created just before the backup was made. The format is

RECOVER TABLE *table-name*

where *table-name* is the name of the table you want to recover.

Once you recover the table, use the DROP AUDIT statement to remove the contents of the audit trail file. Then run the CREATE AUDIT statement to start a new audit trail file and back up the table.

The backup copy must be in the exact state the table was in when the audit trail was started. If this is not the case, the unique record identifiers of the rows in the table will not match their counterparts in the audit trail file. **INFORMIX-SQL** displays a warning message indicating that audit trail recovery deletions could not be made.

Strategies When Using Audit Trails

Since audit trails affect performance, storage, and the number of open files allowed, you should carefully consider when to start and stop audit trails on your database files. Consider this information in light of the prior comparison of audit trails and the transaction log when deciding which tool to use to assist with data integrity.

- Audit trails slow **RDSQL** slightly, because each modification to the table is automatically recorded in the audit trail file. Transactions place a greater demand on the system by recording modifications to all tables in the database.

- With large tables, or tables that are updated frequently, the audit trail file can become quite large. The transaction log grows even more quickly.

- The existence of audit trails decreases the number of database files that RDSQL can open simultaneously. Without an audit trail, each open database file requires that two system files (the **.dat** and **.idx** files) be opened. With an audit trail, a third system file is opened whenever the table is used.

- An audit trail allows you to recover a single table, but it provides no protection against a complex insert or update that fails to run to completion. Transactions do give you the ability to ROLLBACK the database to the last committed transaction and discard the most recent changes.

For many users, an audit trail placed on the most important table in the database is sufficient protection. The tradeoff in system performance is nominal. Create a database with transactions when you must be certain that multiple tables remain in balance, or when you must have the option to ROLLBACK the database and discard changes.

Locking

RDSQL uses locking to prevent different users from executing conflicting operations on the same data. Without locking, for example, two users could update the same row at the same time. In this situation, the computer's memory contains two different versions of the row, the one being updated by user A and the one being updated by user B. Without some method of *concurrency control*, the user whose row is the last one actually written to the file "wins" and overwrites the other user's changes.

RDSQL provides two levels of locking to prevent such an occurrence:

- Table-level or file-level locking
- Row-level or record-level locking

The following sections describe each level of locking and the methods for employing it.

Table-level Locking

You can use table-level locking to lock an entire table and prevent others from altering, or even seeing, rows in that table.

You may want to use this form of locking during batch operations that affect every row in a table, for example. If the operations must be completed as a single transaction, you may find it more efficient to lock the entire table at the beginning of the transaction. Normally, under transactions, RDSQL locks each row as it is updated, deleted, or inserted. If you lock the entire table, however, RDSQL will not use this row-level locking because it is unnecessary. As a result, you are not likely to reach the limits that your operating system may place on the number of rows that can be locked at one time.

You may also wish to lock a table when you want to prevent others from seeing the data in the table for a time.

RDSQL locks a table when you execute a LOCK TABLE statement. The syntax for this statement is

LOCK TABLE *table-name* IN {SHARE | EXCLUSIVE} MODE

where *table-name* is the name of the table to be locked.

If you lock the table IN SHARE MODE, other users can select data from the table or view rows in PERFORM, but they cannot insert, delete, or update rows in the table. If you lock the table IN EXCLUSIVE MODE, other users cannot access the table until you execute an UNLOCK TABLE statement.

Because locking an entire table prevents others from adding or altering data in the table, you should use this feature sparingly. Only lock the entire table when row-level locking (as described in the following section) does not suffice.

Row-level Locking

You can use row-level locking to lock individual rows in a table rather than the entire table. RDSQL handles row-level locking automatically, but you can use indirect methods to lock a row or set of rows explicitly. RDSQL provides two forms of row-level locking:

- Group locking
- Individual locking

Locking Groups of Rows

Group locking locks an entire set of rows until some operation is completed. When the operation is finished, the entire group of rows is freed. RDSQL uses this form of locking during a set of operations that must be completed as a single transaction, for example. In this situation, you do not want others to alter rows that you have changed until you are sure that the transaction has been successfully completed.

If you have created a database with transactions, you can use RDSQL transaction statements to lock rows in this manner. Begin the transaction with a BEGIN WORK statement. After this statement, RDSQL locks every row affected by an UPDATE, DELETE, or INSERT statement. The entire set of rows remains locked until you execute a COMMIT WORK or ROLLBACK WORK statement.

You should be careful that the statements within the transaction do not lock too many rows before the COMMIT WORK or ROLLBACK WORK statement, since you may exceed the maximum number of locks available in your operating system. If you are locking a large number of rows, you may want to divide the operation into two or more smaller transaction sessions, if possible, or lock the entire table, as described in the previous section.

In addition to explicitly using transactions, you lock a group of rows when you execute a single RDSQL INSERT, UPDATE, or DELETE statement that affects multiple rows. When you create a database with transactions, RDSQL treats individual statements as single transactions, even if no BEGIN WORK statement has been executed. For example, an UPDATE statement that affects a number of rows locks the rows until the entire update is completed.

Locking Individual Rows

RDSQL automatically locks individual rows of a table when you execute an UPDATE statement or enter **Update** mode in PERFORM. If you want explicitly to lock an individual row in a table for a period of time, you can query for the row in PERFORM and then enter **Update** mode. The row remains locked until you exit from **Update** mode by either pressing ESC or by pressing the program interrupt key.

Each of the above methods is useful in a particular set of circumstances. Together, these techniques should meet the demands of any situation in which you require locking facilities.

Wait for Locked Row

If another user locks a table in EXCLUSIVE MODE and you attempt to alter, to delete, or even to read a row in the table, RDSQL returns an error code.

If another user locks a row in a table at the row level and you attempt to alter or delete that row (or even examine it with SELECT statement FOR UPDATE), RDSQL returns an error stating that the row is locked. If you would prefer that RDSQL wait on any locked row until the competing process unlocks it, you can execute the SET LOCK MODE TO WAIT statement. From then on, your request will wait until RDSQL unlocks the requested row and you will not receive an error message. If for some reason the competing process fails without unlocking the row, your process will be deadlocked. Consequently, you should use the SET LOCK MODE statement with due caution.

The wait-for-lock feature applies only on multi-user systems that support record-level locking.

NULL Values

The basic purpose of introducing NULL values in a database is to indicate when no value has been assigned to a particular column in a particular row of a table. Your 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 into other columns.

As an example, consider entering data for a bank customer who is requesting a loan. If the customer, Mr. Sutherland, is not employed, the **employer** column in the **client** table will have no entry for this customer. This CHAR column will have the value NULL. The **hire_date** column is meaningless if Mr. Sutherland is not employed. There is no appropriate date to enter; the value is NULL.

Default Values

In RDSQL, the default value for a column is NULL. RDSQL makes a distinction between zero and NULL for numeric values and between blanks and NULL for character values. You do not need to know how RDSQL implements the value NULL to make use of it.

By definition, type SERIAL columns can never contain the NULL value. Columns of type SERIAL always contain integers greater than or equal to one.

You can disallow NULL values in a column of any type by using the NOT NULL clause in the CREATE TABLE statement. RDSQL will prevent a NULL from being entered into any column that is declared NOT NULL.

You cannot include a NOT NULL clause in an ALTER TABLE statement when you add a new column. This is because RDSQL enters a NULL value into the new column for all existing rows.

A column for which you create a unique index can have, at most, one NULL.

Default Values in RDSQL Version 1.10

In the 1.10 version of **RDSQL**, when no value was provided for a column entry in a row of a table, **RDSQL** enters a blank for type CHAR columns, zeroes for numeric columns, and a very large negative value for type DATE columns. Since zero could be an acceptable value for a numeric column (for example, the value for a MONEY column), there is no way to distinguish an unknown value from zero.

To use existing 1.10 databases with Version 2.1 of **INFORMIX-SQL**, you must execute the **dbupdate** utility. Appendix I describes this utility and how you can avoid using NULL values.

The NULL in Expressions

If any value that participates in an arithmetic expression is NULL, the value of the entire expression is NULL. For example, consider the following query:

```
select order_num, ship_charge/ship_weight
  from orders
  where order_num = 1023;
```

If either **ship_charge** or **ship_weight** is NULL (because the order with number 1023 is new and the shipping charge or shipping weight has not yet been determined) the value returned for **ship_charge/ship_weight** will also be NULL.

The situation is different when you use one of the aggregate functions (see the end of this chapter for a description of the aggregate functions). COUNT(*) counts all rows, even if the value of every column in the row is NULL. COUNT(DISTINCT *column-name*), AVG, SUM, MAX, and

MIN ignore rows with NULL values for the column in their argument and return the appropriate value based on the rest of the rows. If a column contains only NULL values, however, COUNT(DISTINCT *column-name*) returns a value of zero, and the other four aggregate functions return NULL for that column.

The NULL in Boolean Expressions

To incorporate NULL values into Boolean expressions, it is necessary to enlarge the number of truth values from simply true and false to include unknown. If one of the expressions of a Boolean expression is NULL, the truth value of the expression is unknown. For example, the Boolean expression

```
ship_charge/ship_weight < 5.0
```

has the truth value unknown for the order in the example shown previously.

If you combine Boolean expressions using the operators AND, OR, and NOT, the following tables give the resulting truth value (where T corresponds to true, F to false, and ? to unknown).

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT	
T	F
F	T
?	?

The NULL in WHERE Clauses

If the Boolean Condition in a WHERE clause evaluates to unknown for a particular row, **RDSQL** treats the search condition as not satisfied and does not select or modify that row.

Consider this clause

```
where ship_charge/ship_weight < 5
   and order_num = 1023
```

The row where **order_num** = 1023 is the row where **ship_weight** is NULL. Since **ship_weight** is NULL, **ship_charge/ship_weight** is also NULL and the truth value of **ship_charge/ship_weight** > 5 is unknown. Since **order_num** = 1023 is true, the AND truth table shown previously states that the truth value of the entire search condition is unknown. Consequently, that row will not be chosen. If the search condition used an OR in place of the AND, the search condition would be true.

You can select (or reject) rows containing NULL values with a new type of search condition:

---

column IS [NOT] NULL

You must use the IS keyword. You are not permitted to write the condition as follows.

---

column = NULL (Incorrect)
column != NULL (Incorrect)

If you perform a join between two tables using the WHERE clause

```
where column1 = column2
```

RDSQL does not select the rows where either **column1** or **column2** is NULL. In particular, no row is returned if both **column1** and **column2** are NULL. This is merely a special case of the more general rule that Boolean expressions containing NULL values have an unknown truth value.

Similarly, if a subquery returns NULL, the outer search condition evaluates to unknown. Subqueries are discussed later in this chapter in "The SELECT Statement" section.

The NULL in ORDER BY Clauses

For the purpose of sorting rows using the ORDER BY clause, the NULL value is treated as being less than a non-NULL value. When the ordering is ASC, the NULL values come first; when the ordering is DESC, the NULL values come last.

The NULL in GROUP BY Clauses

RDSQL treats each row containing a NULL value in a column included in a GROUP BY clause as in a separate group. This is consistent with the fact that each NULL value is really unknown. There is no reason to assume that all NULL values are the same.

The NULL Keyword in INSERT and UPDATE Statements

When you execute the INSERT statement, RDSQL inserts the NULL value into all columns for which you do not provide a value and/or for all columns not listed explicitly. Since the *value-list* of the INSERT statement must be the same length as the *column-list*, you can use the keyword NULL to indicate that a column in *column-list* should be assigned a NULL value.

```
insert into orders (order_num, order_date,
   customer_num) values (0, null, 123);
```

All other columns in the **orders** table will be filled with NULL values.

Similarly, you can use the NULL keyword to modify a column value when using the UPDATE statement. For a customer whose previous address required two address lines, but now requires only one, you would use the following entry:

```
update customer
   set address1 = "123 New Street",
       address2 = null,
       city     = "Palo Alto",
       zip      = "94303"
 where customer_num = 134;
```

No NULL Databases

You may want to avoid dealing with NULL values and their three-valued logic. You can do this if you carefully adhere to the following rules:

- When creating new tables, define all columns as NOT NULL.
- In all form specification files, add the WITHOUT NULL INPUT clause to the DATABASE section.
- In form specification files, specify all **displayonly** fields as NOT NULL.

Views

Views are constructs on a database that allow you to complete the following tasks:

- Provide different users with different windows (called "views") on the data in the database. A single view may include columns from different tables or may show values that are derived from values in the tables. A view has a name and looks to a user as if it is a table. The user may query a view, for example, using the same syntax as though the view were a table in the database.

- Limit access to sensitive data by allowing users to see only aggregate information. With the GRANT and REVOKE statements, you can prevent a user from seeing any salary data in a personnel table. For example, with a view you can allow the user to see average salaries in various groups, but still protect the individual salary data.

- Permit users to update, to insert, and to delete data in the database as though the data were organized as it appears in a view. You can also examine the changes made in a real table of the database through a view.

Views are dynamic windows into the database and are not static snapshots. They differ in this respect from a temporary table created by the INTO TEMP clause of a SELECT statement or the CREATE TEMP TABLE statement. Such temporary tables only show you the state of the database when the table was created. Views reflect the current state of the database.

Although views appear to the user to be tables in the database, there are several important ways in which they differ. You cannot use a view in place of a table in a form specification file. You cannot create an index on a view. There are conditions under which you cannot update or modify the data perceived through a view. An obvious case occurs when the "column" seen in a view is really an expression generated from actual database tables. In general, there is no way to determine the

appropriate change in the underlying columns involved in such an expression should you wish to change the value of the "column."

The next sections describe how to create and delete views, how to query the database through views, how to modify the database through a view, and how to set up permissions for a view.

Creating and Deleting Views

You must use the CREATE VIEW statement to create a view. A view is determined by a SELECT statement that returns the "table" that defines the view. (Note: A UNION operator cannot be used in the definition of a view. See the end of this chapter for the definition of the UNION operator.)

The SELECT statement used in creating a view is stored in the **sysviews** system catalog. When you subsequently make reference to a view in another statement, **RDSQL** performs the defining SELECT statement in executing the new statement.

You can use the same column names as in the underlying table for the view or you can assign new names. When a column in a view is the evaluation of an expression or is not unique (because, for example, you have included all the columns of a join, and there is a column name that is common to the joined tables), you must supply new names. These column names are stored in **syscolumns** system catalog, along with the column names of regular tables.

You delete a view by executing the DROP VIEW statement. When you drop a view, you also drop all views that were defined in terms of that view.

Querying through Views

You can make queries involving views exactly as though they were tables in the database. If possible, **RDSQL** first combines the view-defining SELECT statement with the query to create a new SELECT statement and then executes the new statement. Otherwise, it creates the view as a temporary table and applies the query to the table. **RDSQL** may detect errors during either of these phases.

Modifying through Views

Like querying through views, you may use the INSERT, UPDATE, and DELETE statements with views. **RDSQL** combines the view-defining SELECT statement with the view-referring statement and then executes it. The following restrictions apply to modifying tables through a view:

- You cannot modify the database through a view if the view definition involves joins, the GROUP BY clause, the DISTINCT keyword, or an aggregate function. If any of these features is present in the view definition, the creator of the view does not receive INSERT, DELETE, or UPDATE privileges on the view. You may, however, define a view using a subquery that refers to another table and can often circumvent the restriction on joins. (See the section "Data Constraints Using Views," later in this chapter.)

- You can build a form based on a view as long as the columns contributing data to the view belong to only one table. Aggregate data is not allowed.

- A view column may be updated only if it is derived directly from a column in a table of the database and not as a result of an expression. Expression-derived columns are called "virtual" columns. You cannot INSERT rows through a view that contains virtual columns, although you may DELETE a row that contains a virtual column.

- You may not execute the ALTER TABLE, CREATE INDEX, or UPDATE STATISTICS statements on a view. You do receive the benefit of existing indexes on columns in the underlying tables.

- You may use an INSERT statement on a view that shows only a portion of an underlying table. When you do so, the unspecified columns of the underlying table receive NULL values. If one of the unspecified columns does not permit NULL values, **RDSQL** does not permit you to INSERT to the view.

- If you drop a column used in the creation of the view, **RDSQL** issues an error if you subsequently refer (other than by issuing a DROP VIEW statement) to the view.

- Unless you create the view with a WITH CHECK OPTION clause, it is possible to INSERT or UPDATE data into a database through a view that does not satisfy the constraints on the view. A row inserted or updated in this manner will no longer be accessible through the view.

  For example, a view could be created that allows the user access only to customers from Palo Alto. If, when using the view, a new row is created with a customer from Menlo Park, the user cannot select the row through the view. If the column **city** on an existing row is updated to "Menlo Park," the row disappears from the view. The WITH CHECK OPTION clause on the CREATE VIEW statement causes **RDSQL** to reject an UPDATE or INSERT that violates the restrictions of the view.

- You must be careful when you UPDATE a table through a view that may contain duplicate rows. Duplicate rows can occur in a view even if the underlying table has unique rows.

  For example, suppose a view is defined on the **items** table and contains only the columns **order_num** and **total_price**. The view will contain duplicate rows if two items from the same order have the same total price. If you update the **total_price** through the view, you have no way of knowing which item you have increased.

Privileges with Views

When you create a view, you receive the same privileges that you had on the underlying tables. If you have these privileges with the GRANT OPTION, you may grant privileges on your view to other users. (See the GRANT statement in the last section of this chapter.)

If the view is built on more than one table, you can only have the SELECT privilege. Multi-table views do not permit you to INSERT, DELETE, or UPDATE. You must have the SELECT privilege on all of the columns from which a multi-table view is derived to have the SELECT privilege on the entire view. If, as a result of these restrictions, you have no privileges on a view, the CREATE VIEW statement returns an error.

Data Constraints Using Views

The purpose of data constraints is to ensure that all data entered into the database satisfies pre-assigned limitations. **PERFORM** controls data entry though screen forms by using the INCLUDE attribute to list values and ranges of values permitted for a column. (See Chapter 3, "The FORMBUILD Transaction Form Generator.") In this case, however, the list of allowed values is static and is dependent only on the designated column.

It is often desirable to allow value ranges to be dynamically defined, based on the values in other columns or even in other tables. The existence of views and, specifically, the WITH CHECK OPTION clause permits the owner of the table to control the entry of data into the database. This is most easily demonstrated with an example taken from the **stores** database.

Suppose you want to ensure that no item

- Has a value of more than $20,000
- Is for stock that does not exist

The first step is to create the following view:

```
create view safe_items as
    select * from items
        where total_price < 20000 and
            exists (select stock_num, manu_code
        from stock
        where stock.stock_num=items.stock_num
        and stock.manu_code=items.manu_code)
    with check option;
```

If you do all data entry and data modification through the **safe_items** view, RDSQL will reject all data that does not meet the requirements of the WHERE clause. Because of the dynamic nature of views, the view will only contain rows corresponding to current stock items if you change the **stock** table by adding rows corresponding to new stock items or by deleting old ones.

By extending the WHERE clause, this example can be expanded to cover very general data-constraint needs.

Outer Joins

An outer join between two tables treats the two tables asymmetrically. One of the tables (sometimes called a "preserved" table) is dominant and the other table is subservient. If the subservient table has no rows satisfying the join condition, the outer join attaches a row of NULL values to the row of the dominant table before projecting the desired columns. To illustrate, let **a** be a column from **tab1** and **b** a column in **tab2**. Further, let the values in the two tables be as shown in the following display:

tab1.a	tab2.b
2	4
3	2
5	6
	5

RDSQL syntax requires that the subservient table in an outer join be preceded by the keyword OUTER in the FROM clause. The following SELECT statement contains an outer join between **tab1** (the dominant table) and **tab2** (the subservient table):

```
select a, b
    from tab1, outer tab2
    where a = b;
```

The resulting table has the following two columns:

a	b
2	2
3	-
5	5

Every value for **a** is joined with only those values of **b** that match **a**. When there is no value in column **b** that satisfies the join condition, a NULL value (shown here as -) is substituted.

A WHERE clause is required in the case of outer joins and must specify a relationship between the two tables. (See Appendix L for more information about outer joins.)

Table Access by ROWID

The keyword ROWID can be used in **RDSQL** statements to refer to the **C-ISAM** record number associated with a row in a database table. ROWID can be thought of as a hidden column in every table. When you use the following statement

```
select * from table
```

the implied list of columns does *not* include ROWID. On the other hand, you can use the syntax

```
select rowid, * from table-name;
```

to get the ROWID value for each row.

ROWID can also be used in WHERE clauses to select rows based on their **C-ISAM** record number. This feature is useful when there is no other unique column in a table:

```
select rowid, * from table-name
where rowid > 7;
```

If a row is deleted from the table, its ROWID can be assigned to a new row.

TODAY and USER Functions

RDSQL provides functions to allow you to include today's date and the user name in a statement. TODAY always returns the system's date. USER returns a string containing the current user's name. On UNIX systems, this is the login name, while on multi-user DOS systems this is the machine name.

You can use these functions wherever you may use a constant. For example, if you wish to retrieve the rows that you have inserted into a table, you must first define a column to contain the USER name. Whenever you insert into the table, you may use the USER function as follows:

```
insert into table values (...,user,...);
```

You may then retrieve the rows that you entered with a select statement, as follows:

```
select * from table
   where user_col = user
```

You may use the TODAY function in the same way. To insert the system date into a table, you can use the following statement:

```
insert into table values (...,today,...);
```

To retrieve all rows with today's date from a table, the following statement can be used.

```
select * from table
   where date_col = today
```

Examples

The first example retrieves the ROWID and all rows from the **userlist** table where the value in the **uname** column matches the value returned from the USER function.

The second example outputs to the *userfile* file all rows from the **employee** table where the value in the **fname** column matches the value returned from the USER function.

```
select rowid, * from userlist
   where uname matches user;

output to userfile
   select * from employee where fname matches user;
```

RDSQL Statements

The RDSQL statements below are described in detail on the following pages. They are presented in alphabetic order. Because the SELECT statement and its clauses (FROM, WHERE, GROUP BY, HAVING, ORDER BY, and INTO TEMP) require special attention, they are described in a later section.

ALTER INDEX
ALTER TABLE
BEGIN WORK
CHECK TABLE
CLOSE DATABASE
COMMIT WORK
CREATE AUDIT
CREATE DATABASE
CREATE INDEX
CREATE SYNONYM
CREATE TABLE
CREATE VIEW
DATABASE
DELETE
DROP AUDIT
DROP DATABASE
DROP INDEX
DROP SYNONYM
DROP TABLE
DROP VIEW
GRANT
INFO
INSERT
LOAD
LOCK TABLE
OUTPUT
RECOVER TABLE
RENAME COLUMN
RENAME TABLE
REPAIR TABLE
REVOKE
ROLLBACK WORK
ROLLFORWARD DATABASE
SELECT
SET LOCK MODE
START DATABASE
UNLOAD
UNLOCK TABLE
UPDATE
UPDATE STATISTICS

ALTER INDEX

Overview

Use the ALTER INDEX statement to cluster a table in the order of an existing index or to release an index from the clustering attribute.

Syntax

ALTER INDEX *index-name* TO [NOT] CLUSTER

Explanation

ALTER INDEX	are required keywords.
index-name	is the **RDSQL** identifier of an existing index.
TO	is a required keyword.
NOT	is an optional keyword.
CLUSTER	is a required keyword.

Notes

1. The TO CLUSTER option causes **RDSQL** to reorder the rows in the physical table to agree with the order of *index-name*. Reordering causes the entire file to be rewritten and requires sufficient disk space to maintain two copies of the table. The process may take a long time.

2. Since there may be only one clustered index per table, you must use the NOT option to release the cluster attribute from one index before assigning it to another. The NOT option does not affect the physical table; it merely drops the cluster attribute on *index-name* from the system catalogs.

3. When **RDSQL** executes ALTER INDEX with the TO CLUSTER option, it locks the table in EXCLUSIVE mode. If some other process is using the table to which *index-name* belongs, **RDSQL** cannot execute ALTER INDEX with the TO CLUSTER option and returns an error.

4. If you execute this statement within a transaction, it cannot be rolled back. It may, however, be rolled forward.

5. Over time you can expect the benefit of an earlier clustering to disappear. You can recluster the table by issuing another ALTER INDEX TO CLUSTER statement on the clustered index.

6. You do not need to drop a cluster index before issuing another ALTER INDEX TO CLUSTER statement on a currently clustered index.

Examples

The following example creates two indexes on the **orders** table and clusters the physical table in ascending order on the **customer_num** column. Later, the example clusters the physical table in ascending order on the **order_num** column.

```
create unique index ix_ord
       on orders (order_num);
create cluster index ix_cust
       on orders (customer_num);
...

alter index ix_cust to not cluster;

alter index ix_ord to cluster;
```

Related Statements

CREATE INDEX

ALTER TABLE

Overview

Use the ALTER TABLE statement to add a column to a table, to delete a column from a table, or to modify the data type of a column.

Syntax

```
ALTER TABLE table-name
        {ADD (newcol-name newcol-type
                [BEFORE oldcol-name], ...)
        |DROP (oldcol-name, ...)
        |MODIFY (oldcol-name newcol-type [NOT NULL] , ...)}, ...
```

Explanation

ALTER TABLE	are required keywords.
table-name	is the name of an existing table.
ADD	is a keyword you use to add a column to *table-name*.
newcol-name	is the name of the column you want to add.
newcol-type	is either the data type of the column you are adding or of the column whose data type you are modifying.
BEFORE	is an optional keyword you use to indicate where you want *newcol-name* placed in the list of columns. The default is at the end of the list of columns.
oldcol-name	is the name of an existing column.

DROP	is a keyword you use to drop a column from *table-name*.
MODIFY	is a keyword you use to change the data type of an existing column.
NOT NULL	are optional keywords.

Notes

1. You may use one or more of the ADD clause, the DROP clause, or the MODIFY clause and may place them in any order. Use a comma to separate the clauses. The actions are performed in the order specified. If any of the actions fail, the entire operation is canceled.

2. You may not add a SERIAL column to a non-empty table.

3. You cannot use a NOT NULL clause in an ALTER TABLE statement when you add a new column, since **RDSQL** inserts a NULL value into the new column for all existing rows.

4. If a column is currently designated as NOT NULL, you can modify it to allow NULL values by specifying the same column name and data type in the MODIFY clause of an ALTER TABLE statement. If a column allows NULL values but currently does not contain any NULLs, you can change it to a NOT NULL column by specifying the same column name and data type in the MODIFY clause along with the NOT NULL keywords.

5. You must own *table-name*, have DBA privilege, or be granted ALTER permission to use ALTER TABLE.

6. Do not use the ALTER TABLE statement within a transaction; it cannot be rolled back.

Examples

```
alter table items
    add (item_weight decimal(6,2)
        before total_price);

alter table items
    drop (total_price);

alter table items
    modify (manu_code char(4));
```

Related Statements

RENAME COLUMN, RENAME TABLE

BEGIN WORK

Overview

Use the BEGIN WORK statement to start a transaction (a sequence of database operations that are terminated by the COMMIT WORK or ROLLBACK WORK statement).

Syntax

BEGIN WORK

Explanation

BEGIN WORK are required keywords.

Notes

1. See the "Transactions" section, earlier in this chapter, for a full description of transactions.

2. Each row affected by an UPDATE, DELETE, or INSERT statement during a transaction is locked and remains locked throughout the transaction. A transaction that contains a large number of such statements, or that contains statements affecting a large number of rows, may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If you encounter this error, you can lock the entire table after beginning the transaction. You must issue the BEGIN WORK statement before issuing the LOCK TABLE statement.

   See the "Locking" section in this chapter for a more detailed description of table-level and row-level locking in **INFORMIX-SQL**.

Related Statements

COMMIT WORK, ROLLBACK WORK

CHECK TABLE

Overview

Use the CHECK TABLE statement to compare a table with its indexes to determine if they match. Use this statement when you think that the data or the indexes might be corrupted because of a power failure, computer crash, or other abnormal program stoppage while you are using INFORMIX-SQL. CHECK TABLE uses the **bcheck** utility. See Appendix G for a full description of **bcheck**.

Syntax

CHECK TABLE *table-name*

Explanation

CHECK TABLE	are required keywords.
table-name	is the name of the database table and associated indexes you want to check.

Notes

1. You must own *table-name* or have DBA database privilege to use CHECK TABLE.

2. You cannot use CHECK TABLE on the system catalog **systables**.

Examples

```
check table customer;
```

Related Statements

REPAIR TABLE

CLOSE DATABASE

Overview

Use the CLOSE DATABASE statement to close the current database and to leave the program with no current database.

Syntax

CLOSE DATABASE

Explanation

CLOSE DATABASE are required keywords.

Notes

1. Following the CLOSE DATABASE statement, the only legal **RDSQL** statements are CREATE DATABASE, DATABASE, DROP DATABASE, and ROLLFORWARD DATABASE.

2. You must issue a CLOSE DATABASE statement before you drop the current (or only remaining) database.

Related Statements

CREATE DATABASE, DROP DATABASE, ROLLFORWARD DATABASE

COMMIT WORK

Overview

Use the COMMIT WORK statement to commit all modifications made to the database since the last BEGIN WORK statement.

Syntax

```
COMMIT WORK
```

Explanation

COMMIT WORK are required keywords.

Notes

1. Use the COMMIT WORK statement when you are satisfied that the changes to the database made since the last BEGIN WORK statement are what you want.

2. COMMIT WORK releases all row and table locks.

3. See the "Transactions" section, earlier in this chapter, for details.

Related Statements

BEGIN WORK, ROLLBACK WORK

CREATE AUDIT

Overview

Use the CREATE AUDIT statement to create an audit trail file and to start writing the audit trail.

Syntax

CREATE AUDIT FOR *table-name* IN "*pathname*"

Explanation

CREATE AUDIT FOR	are required keywords.
table-name	is the name of the table for which you want to create an audit trail file.
IN	is a required keyword.
pathname	is the full pathname for the audit trail file. It must be contained in quotation marks.

Notes

1. You create audit trails to keep a record of all modifications of a table. An audit trail is a complete history of all additions, deletions, and updates to the table. **RDSQL** can use the audit trail to reconstruct the table from a backup copy made at the time the audit trail is created. (See the RECOVER TABLE statement.) See the section about "Audit Trails" in this chapter for more information.

2. If an audit trail file with the same pathname already exists, the CREATE AUDIT statement does nothing. If an audit trail file for the same table exists with a different pathname, **RDSQL** displays an error message.

3. Make a backup copy of your database files as soon as you run the CREATE AUDIT statement, but before you make any further changes to the database. (See RECOVER TABLE for an example.) If possible, put the audit trail file on a different physical device from the one that holds your data, so that a failure of one does not damage the data on the other.

4. Audit trails slow **RDSQL** very slightly because each alteration of the database is recorded in the audit trail file, as well as in the database files.

5. You must own *table-name* or have DBA status to use the CREATE AUDIT statement.

6. You must set execute permission for all directories below **root** in *pathname* for each class of user (owner, owner's group, and public) that accesses your database.

Examples

On a UNIX system:

```
create audit for orders in "/dev/safe";
```

On a DOS system:

```
create audit for orders in "\KEEP\SAFE";
```

Related Statements

DROP AUDIT, RECOVER TABLE

CREATE DATABASE

Overview

Use the CREATE DATABASE statement to create a new database. **RDSQL** will create the system catalogs that will contain the data dictionary describing the structure of the database. The database you create automatically becomes the current database.

Syntax

CREATE DATABASE *database-name* [WITH LOG IN "*pathname*"]

Explanation

CREATE DATABASE	are required keywords.
database-name	is the name you want to assign to the database.
WITH LOG IN	are optional keywords.
pathname	is the full pathname of the transaction log file. *pathname* must be enclosed in quotation marks.

Notes

1. **RDSQL** creates a subdirectory in the current directory with the name *database-name.dbs*. All of the system catalogs, data, and index files will be placed in this subdirectory, except for tables that you explicitly instruct **RDSQL** to create elsewhere. If you want the database in a directory other than the current directory, you must exit from **INFORMIX-SQL** change directories, and return to

INFORMIX-SQL before issuing the CREATE DATABASE statement.

2. A database name can be up to eight (DOS) or ten (UNIX) characters in length and can contain only letters, digits, and underscores(_). The first character must be a letter. If you store more than one database in a single directory, the database names must be unique.

3. For a user to have access to a database on UNIX systems, the user must have execute (search) permission for each directory in the full pathname of *database-name.dbs*, as well as appropriate database privileges (see the GRANT statement).

4. See Appendix K for a detailed description of the system catalogs.

5. The WITH LOG IN clause creates a transaction log file. Without this file, you may not use the BEGIN WORK, COMMIT WORK, or the ROLLBACK WORK statements. You can use the START DATABASE statement to assign a log file to an existing database. See the "Transactions" section, earlier in this chapter, for further details.

6. You can also use the **Database** option on the INFORMIX-SQL Menu to create a database. Using the menu option has the same effect as running the CREATE DATABASE statement within RDSQL.

7. **dbschema** is a utility that produces the RDSQL statements used to create all or part of a database. See Appendix D of this manual for a detailed discussion of this utility.

8. You can determine the location of the log file for the current database by running the following SELECT statement:

```
SELECT dirpath FROM systables
WHERE tabtype = "L";
```

Examples

On a UNIX system:

```
create database stores
    with log in "/s/log/stores.log";
```

On a DOS system:

```
create database stores
    with log in "\LOG\STORES.LOG";
```

Related Statements

DROP DATABASE, GRANT, START DATABASE

CREATE INDEX

Overview

Use the CREATE INDEX statement to create an index for one or more columns in a table and optionally to cluster the physical table in the order of the index. When there is more than one column listed, the concatenation of the set of columns is treated as a single composite column for the purpose of indexing.

Syntax

CREATE [UNIQUE | DISTINCT] [CLUSTER] INDEX *index-name*
ON *table-name* (*column-name* [ASC | DESC], ...)

Explanation

CREATE INDEX	are required keywords.
UNIQUE	is a keyword you use to prevent duplicate entries in the column or composite column to which the index applies.
DISTINCT	is a keyword that is synonymous with UNIQUE.
CLUSTER	is an optional keyword that causes the physical table to be ordered according to the order of the index.
index-name	is the RDSQL identifier you want to assign to the index. You must assign a different identifier to each index in the database.
ON	is a required keyword.

table-name	is the name of the table containing the column or columns you want to index.
column-name	is the name of a column you want to index. To create an index that applies to several columns, enter a list of column names, separated by commas. All the columns you specify must belong to the same table.
ASC	is a keyword that specifies an index that RDSQL maintains in ascending order. ASC is the default.
DESC	is a keyword that specifies an index that RDSQL maintains in descending order.

Notes

1. When RDSQL executes the CREATE INDEX statement, it locks *table-name* in EXCLUSIVE mode. If another process is using *table-name*, RDSQL cannot execute CREATE INDEX and returns an error.

2. You may include up to eight columns in a composite index.

3. The total length of all columns indexed in a single CREATE INDEX statement cannot exceed 120 bytes.

4. See the section "Indexing Strategy," earlier in this chapter, for a discussion of indexing strategy.

5. The CREATE CLUSTER INDEX statement fails if a CLUSTER index already exists.

6. Only one index on a particular sequence of columns is allowed (ASC and DESC are considered to provide uniqueness).

7. If you use the CREATE INDEX statement in a transaction, it cannot be rolled back. It may, however, be rolled forward.

Examples

```
create unique index i_ordnum
    on orders (order_num);

create cluster index i_ordnum2
    on orders (order_num desc);
```

Related Statements

ALTER INDEX, DROP INDEX

CREATE SYNONYM

Overview

Use the CREATE SYNONYM statement to provide an alternative name for a table or view.

Syntax

CREATE SYNONYM *synonym* FOR *table-name*

Explanation

CREATE SYNONYM	are required keywords.
synonym	is an **INFORMIX-SQL** identifier.
FOR	is a required keyword.
table-name	is the name of a table or view.

Notes

1. A synonym is effective only for the user who creates it.

2. A user has no privileges under a synonym that were not granted for the table to which the synonym applies.

3. Once created, a synonym persists until the owner of the synonym executes the DROP SYNONYM statement. This property distinguishes the synonym from the alias that you can use in the FROM clause of a SELECT statement. The alias persists only until the completion of the SELECT statement.

4. Do not use the CREATE SYNONYM statement in a transaction; it cannot be rolled back.

5. The synonym name cannot be the same as a table in the database.

Examples

```
create synonym c for customer;
```

Related Statements

DROP SYNONYM, SELECT

CREATE TABLE

Overview

Use the CREATE TABLE statement to create a new table in the current database.

Syntax

CREATE [TEMP] TABLE *table-name*
(*column-name datatype* [NOT NULL], ...)
[IN "*pathname*"]

Explanation

CREATE TABLE	are required keywords.
TEMP	is an optional keyword.
table-name	is the identifier you want to assign to the table. The first 8 (DOS) or 10 (UNIX) characters must be unique within a database.
column-name	is the identifier you want to assign to each column.
datatype	specifies the data type for each column. (See the following list for valid data types.)
NOT NULL	are optional keywords.
IN	is an optional keyword.
pathname	specifies the full pathname in which you want to store the database table, with no extension to the filename. The *pathname* cannot be longer than 64 characters, and must be contained within quotes (").

In UNIX systems, a pathname is of the form

[/directory-name/...]filename

and in DOS systems the form of a pathname is

[\directory-name\...]filename

Valid data types in **RDSQL** follow. (A detailed discussion of data types is contained in Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide.*)

CHAR(*n*)	A character string of length *n* (where 1 <= *n* <= 32,767).
SMALLINT	A whole number from −32,767 to +32,767.
INTEGER	A whole number from −2,147,483,647 to +2,147,483,647.
DECIMAL[(*m*[,*n*])]	A decimal floating-point number with a total of *m* (<= 32) significant digits (precision) and *n* (<= *m*) digits to the right of the decimal point (scale). See the section "Database Data Types," earlier in this chapter, for more details.
SMALLFLOAT	A binary floating-point number corresponding to the "float" data type in the C language. On small machines, **RDSQL** implements SMALLFLOAT as DECIMAL(8).
FLOAT	A binary floating-point number corresponding to the "double" data type in the C language. On small machines, **RDSQL** implements FLOAT as DECIMAL(16).

MONEY[(*m*[,*n*])]	A DECIMAL type number, displayed with leading **$**. MONEY(*m*) = DECIMAL(*m*,2) and MONEY = DECIMAL(16,2). See the section "Database Data Types," earlier in this chapter, for more details.
SERIAL[(*n*)]	A unique sequential integer assigned automatically by **RDSQL**. You may assign an initial value *n*. The default starting integer is 1.
DATE	A date entered as a character string in one of the formats described in the following notes.

Notes

1. Table names must be unique within a database.

2. Column names must be unique within each table, but you can use duplicate names in different tables in the same database.

3. Temporary tables created with the TEMP option last only for the duration of the session (that is until you exit **INFORMIX-SQL**). You may not build a **PERFORM** screen on a temporary table.

4. Users with CONNECT privilege may create temporary tables.

5. The default value for a column is NULL, unless you include the NOT NULL keywords after the data type of the column.

6. If you designate a column as NOT NULL, you must enter a value into this column when performing an INSERT or UPDATE to the table.

7. By default, all users who have been granted CONNECT privilege to the database have all types of access to the new table except ALTER. To further restrict access, use the REVOKE statement to take *all* access away from PUBLIC (everyone on the system); then use the GRANT statement to designate the types of access you want to give to specific users.

8. You can specify no more than one SERIAL column in a table.

9. Enter DATE data types in the sequence of month, day, and year, with any non-numeric character, including a blank, as a separator. Represent the month as the number of the month (January = 1 or 01, February = 2 or 02, and so on). Represent the day as the day of the month (1 or 01, 2 or 02, and so on). The year is stored as a four-digit number (0001 to 9999). If you enter two digits *yy* for the year, **RDSQL** assumes the year is *19yy*.

   The following are all acceptable representations of June 1, 1986: 06/01/86, 6.1.86, and 6-1-1986.

10. The DATE type is actually stored as the integer number of days since December 31, 1899. You can sort DATE columns and make chronological comparisons between two DATE columns.

11. The file space requirements in bytes for each data type are

SERIAL	4
SMALLINT	2
INTEGER	4
SMALLFLOAT	4
FLOAT	8
CHAR(n)	n
DECIMAL(m,n)	1 + m/2
MONEY(m,n)	1 + m/2
DATE	4

   On small machines, SMALLFLOAT requires five bytes, and FLOAT, nine bytes.

12. Do not use the CREATE TABLE statement within a transaction; it cannot be rolled back.

13. If the *pathname* in an IN clause specifies a filename that is different from the *table-name*, always use the *table-name* (rather than the filename) to refer to the table in subsequent **INFORMIX-SQL** statements.

14. The *pathname* in an IN clause can specify any valid directory and is not restricted to the drive or directory that contains the current database. This enables a database to include tables in different disk volumes, or on more than one drive, or on different network nodes, or on virtual drives. Use this feature if your database is becoming too large for your current disk volume.

Examples

The following statement creates the **cust** table and places the database files in the **/u/mirage/fluffy** file:

```
create table cust
    (
    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)
    )
    in "/u/mirage/fluffy";
```

The sequence of statements that creates the **stores** database follows:

```
create database stores;

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)
    );

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
    );

create table items
    (
    item_num          smallint,
    order_num         integer,
    stock_num         smallint,
    manu_code         char(3),
    quantity          smallint,
    total_price       money(8)
    );

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

create table manufact
    (
    manu_code         char(3),
    manu_name         char(15)
    );

create unique index c_num_ix on customer (customer_num);
create index zip_ix on customer (zipcode);
create unique index o_num_ix on orders (order_num);
create index o_c_num_ix on orders (customer_num);
create index i_o_num_ix on items (order_num);
create index st_man_ix on items (stock_num, manu_code);
create unique index stk_man_ix on stock (stock_num, manu_code);
create unique index man_cd_ix on manufact (manu_code);
```

Related Statements

CREATE DATABASE, DROP DATABASE, DROP TABLE, GRANT, REVOKE

CREATE VIEW

Overview

Use CREATE VIEW to create a new view based upon existing tables and views in the database.

Syntax

CREATE VIEW *view-name* [(*column-list*)]
AS *SELECT-statement* [WITH CHECK OPTION]

Explanation

CREATE VIEW	are required keywords.
view-name	is an **RDSQL** identifier.
column-list	is a list of one or more identifiers that name the columns of *view-name*.
AS	is a required keyword.
SELECT-statement	is an **RDSQL** SELECT statement.
WITH CHECK OPTION	are optional keywords.

Notes

1. Except for the statements in the following list, you may use a view in any **RDSQL** statement where you can use a table.

ALTER TABLE	DROP INDEX
ALTER INDEX	LOCK TABLE
CREATE INDEX	RENAME TABLE

The view will behave like a table having the name *view-name* and consisting of the set of rows and columns returned by the *SELECT-statement* each time the RDSQL statement is executed using the view. The view will reflect changes to the underlying tables with one exception. If the view is defined with a SELECT * clause, it will have only the columns in the underlying tables at the time the view is created. New columns added subsequently to the underlying tables using the ALTER TABLE statement will not appear in the view. See the section on "Views" in this chapter for more information.

2. If you do not specify *column-list* for *view-name*, the view will inherit the column names of the underlying tables. If the *SELECT-statement* returns an expression, the corresponding column in the view is called a *virtual* column. You must provide a name for virtual columns. You must also provide a column name when the *select-list* has duplicate column names when the table prefixes are stripped. For example, when both **orders.order_num** and **items.order_num** appear in the *select-list*, you must provide two separate column names to label them in the CREATE VIEW statement.

3. If you must provide names for some of the columns in a view, then you must provide names for all of the columns; that is, the column list must contain an entry for every column appearing in the view.

4. Data types of the columns of the view are inherited from the tables from which they come. Data types of virtual columns are determined from the nature of the expression.

5. *view-name* must be unique among all the tables and views that already exist in the database.

6. You can define a view in terms of other views, except that you must abide by the restrictions on queries listed in the section "Querying through Views," earlier in this chapter.

7. *SELECT-statement* may not have an ORDER BY clause nor a UNION operator.

8. You must have SELECT privilege on all columns from which the view is derived.

9. The WITH CHECK OPTION clause instructs **RDSQL** to ensure that all modifications to the underlying tables made through the view satisfy the definition of the view.

10. Do not use the CREATE VIEW statement within a transaction; it cannot be rolled back.

Examples

```
create view palo_alto as
    select * from customer
        where city = "Palo Alto";
```

Related Statements

CREATE TABLE, DROP VIEW

DATABASE

Overview

Use the DATABASE statement to select an accessible database as the current database.

Syntax

DATABASE *database-name* [EXCLUSIVE]

Explanation

DATABASE is a required keyword.

database-name is the name of the database you want to select.

EXCLUSIVE is an optional keyword.

Notes

1. **RDSQL** ordinarily recognizes only those databases stored in your current directory or in a directory specified in your DBPATH environment variable.

2. The EXCLUSIVE option opens the database in an exclusive mode and allows only the current user access to the database. To allow others access to the database, you must execute the CLOSE DATABASE statement and then reopen the database.

3. You can also use the **Database** option in the INFORMIX-SQL Menu to select a database. Using the menu option has the same effect as running the DATABASE statement within **RDSQL**.

4. Do not use the DATABASE statement within a transaction session; it cannot be rolled back.

Examples

```
database stores;
```

DELETE

Overview

Use the DELETE statement to delete one or more rows from a table.

Syntax

DELETE FROM *table-name* [WHERE *condition*]

Explanation

DELETE FROM	are required keywords.
table-name	is the name of the table from which you want to delete rows.
WHERE	is a keyword.
condition	is a condition of a standard WHERE clause (see the SELECT statement for further information). RDSQL deletes all rows that satisfy the condition in the WHERE clause.

Notes

1. When you create a database with transactions, each RDSQL statement that you execute is treated as a single transaction, even if you do not use the BEGIN WORK and COMMIT or ROLLBACK WORK statements. Because each row affected by the DELETE statement within a transaction is locked for the duration of the transaction, a single DELETE statement that affects a large number of rows locks the rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum

number of simultaneous locks. If this occurs, you may want either to reduce the scope of the DELETE statement or lock the entire table before executing the statement.

See the "Locking" section in this chapter for a more detailed description of table-level and row-level locking in **INFORMIX-SQL.**

Caution! If you omit the WHERE clause while working within the RDSQL Menu, **RDSQL** prompts you to verify that you want to delete all rows from a table. You do not receive a prompt if you run the DELETE statement within a command file.

If you include a WHERE clause that selects all rows in the table, **RDSQL** gives no prompt and deletes all rows.

Examples

```
delete from items
   where order_num = 1034;
```

Related Statements

INSERT, UPDATE

DROP AUDIT

Overview

Use the DROP AUDIT statement to delete an audit trail file.

Syntax

DROP AUDIT FOR *table-name*

Explanation

DROP AUDIT FOR are required keywords.

table-name is the name of the table whose audit trail file you want to delete.

Notes

1. Use the DROP AUDIT statement to remove the old audit trail file when you have completed making a backup of your database files. Use the CREATE AUDIT statement to start a new audit trail, and then back up the table. See the section on "Audit Trails" in this chapter for more information.

2. You must own *table-name* or have DBA status to use the DROP AUDIT statement.

Examples

```
drop audit for orders;
```

Related Statements

CREATE AUDIT, RECOVER TABLE

DROP DATABASE

Overview

Use the DROP DATABASE statement to delete an entire database, including all system catalogs, indexes, and data.

Syntax

DROP DATABASE *database-name*

Explanation

DROP DATABASE are required keywords.

database-name is the name of the database you want to delete.

Notes

1. You must own all the tables in the database or have DBA status to run the DROP DATABASE statement successfully. Otherwise, **RDSQL** issues an error message and executes nothing.

2. The DROP DATABASE statement will not remove the database directory if there are any files in the database directory other than those created for database tables and their indexes.

3. You cannot drop the current database. You must execute the CLOSE DATABASE statement first.

4. Use this statement with caution. If you have DBA privilege, **RDSQL** does not prompt you to verify that you want to delete the entire database.

5. You cannot execute the DROP DATABASE statement within a transaction; it cannot be rolled back.

Examples

```
drop database stores;
```

Related Statements

CREATE DATABASE, CLOSE DATABASE

DROP INDEX

Overview

Use the DROP INDEX statement to delete an index.

Syntax

DROP INDEX *index-name*

Explanation

DROP INDEX are required keywords.

index-name is the name of the index you want to delete.

Notes

1. You must be the owner of the index or have DBA privilege to use the DROP INDEX statement.

2. Do not execute the DROP INDEX statement within a transaction; it cannot be rolled back.

Examples

```
drop index i_ordnum;
```

Related Statements

CREATE INDEX

DROP SYNONYM

Overview

Use the DROP SYNONYM statement to remove a previously defined synonym.

Syntax

DROP SYNONYM *synonym*

Explanation

DROP SYNONYM	are required keywords.
synonym	is an **INFORMIX-SQL** identifier.

Notes

1. You may drop a synonym only if you created it.

2. Do not execute the DROP SYNONYM statement within a transaction; it cannot be rolled back.

Examples

```
drop synonym cust;
```

Related Statements

CREATE SYNONYM

DROP TABLE

Overview

Use the DROP TABLE statement to remove a table, along with its associated indexes and data.

Syntax

DROP TABLE *table-name*

Explanation

DROP TABLE are required keywords.

table-name is the name of the table you want to remove.

Notes

1. When you remove a table, you also delete the data stored in it, the indexes on columns, any synonyms assigned to it, and any authorizations you have granted on the table. You also delete all views based on the table.

2. You cannot drop any of the system catalog tables.

3. You must be the owner of a table or have DBA privilege to use the DROP TABLE statement. Use this statement with caution. **RDSQL** does not prompt you to verify that you want to delete an entire table.

4. Do not execute the DROP TABLE statement within a transaction; it cannot be rolled back.

Examples

```
drop table customer;
```

Related Statements

CREATE TABLE

DROP VIEW

Overview

Use the DROP VIEW statement to remove a view from the database.

Syntax

DROP VIEW *view-name*

Explanation

DROP VIEW are required keywords.

view-name is the identifier of a view.

Notes

1. You can drop only those views that you have created. See the section "Views" in this chapter for more information.

2. When you drop *view-name*, you also drop all views that have been defined in terms of *view-name*.

3. Do not execute the DROP VIEW statement within a transaction session; it cannot be rolled back.

Examples

```
drop view cust1;
```

Related Statements

CREATE VIEW, DROP TABLE

GRANT

Overview

Use the GRANT statement to specify user access privileges to a database or to the tables in a database. Access privileges are relevant only on multi-user systems.

Syntax

GRANT *tab-privilege* ON *table-name* TO {PUBLIC | *user-list*}
[WITH GRANT OPTION]

GRANT *db-privilege* TO {PUBLIC | *user-list*}

Explanation

GRANT — is a required keyword.

tab-privilege — is one or more of the following table-level access types (multiple privileges must be separated by commas):

ALTER	Add or delete columns or modify data types of columns
DELETE	Delete rows
INDEX	Create indexes
INSERT	Insert rows
SELECT [(*cols*)]	Retrieve data from specified columns

	UPDATE [(*cols*)]	Change values in specified columns
	ALL [PRIVILEGES]	All of the above

SELECT and UPDATE take column names as arguments, allowing you to specify columns that the user may select or update. Separate column names with commas.

The keyword PRIVILEGES following ALL is optional.

ON — is a required keyword.

table-name — is the name of the table for which you are granting access privileges.

TO — is a required keyword.

PUBLIC — is the keyword you use to specify access privileges for all users.

user-list — is a list of login names for the users to whom you are granting access privileges. You can enter one login name or a series of login names, separated by commas.

WITH GRANT OPTION — are optional keywords.

db-privilege — is one of the following database-level access types:

CONNECT	Allows access to database tables without permission to create permanent tables and indexes.

RESOURCE	Allows access to database tables with permission to create permanent tables and indexes.
DBA	Allows full database administrator privileges.

Notes

1. You can grant privileges only on tables you create or on tables for which you have been given GRANT OPTION.

2. If you do not enter any column names, the SELECT or UPDATE access that you grant applies to all columns.

3. When you GRANT table-level privileges to another user using the WITH GRANT OPTION phrase, you give that user the power to GRANT the same privileges to another user.

4. The CONNECT privilege allows the recipient to interact with the existing tables of the database with all the table-level access privileges except ALTER and INDEX. The CONNECT privilege alone forbids the recipient from creating tables (except temporary tables). The CONNECT recipient may create views.

5. The RESOURCE privilege includes the CONNECT privilege and adds the permission to create tables.

6. The DBA privilege includes the RESOURCE privilege, as well as the ability to alter system catalogs, to drop, start, and to roll forward the database, and to grant and revoke CONNECT, RESOURCE, and DBA privileges to and from other users.

7. When you create a database, you are the Database Administrator and have DBA privileges.

8. Do not execute the GRANT statement within a transaction; it cannot be rolled back.

9. The most restrictive privileges always take precedence. For example, if you grant RESOURCE privileges to a user but do not grant INDEX privileges at the table level, that user will not be able to create indexes for that table. Similarly, if you grant a user CONNECT privileges and table-level INDEX privileges, that user will still be prevented from using the CREATE INDEX statement.

Examples

When you grant CONNECT privileges to PUBLIC, all table-level privileges, except ALTER, are granted to everyone. The situation is the same as if you ran the following statements for each of the tables in the database:

```
grant all on customer to public;

revoke alter on customer from public;
```

To restrict access privileges on the table level, you must first revoke all privileges and then grant those you want:

```
revoke all on customer from public;

grant all on customer to joe, mary;
grant select (fname, lname, company, city)
    on customer to public;
```

The following statement produces a list of the users and their privileges:

```
select * from sysusers
```

Refer to Appendix K for an explanation of the results of this statement.

Related Statements

REVOKE

INFO

Overview

Use the INFO statement to display

- The names of the tables in the current database
- Column information for a particular table
- Index information for a particular table
- Access privileges for a particular table
- Status information for a particular table

Syntax

```
INFO  {TABLES  | COLUMNS FOR table-name
               | INDEXES FOR table-name
               | [ACCESS | PRIVILEGES] FOR table-name
               | STATUS FOR table-name}
```

Explanation

INFO	is a required keyword.
TABLES	is the keyword you use to display a list of the tables in the current database.
COLUMNS FOR	are the keywords you use to display the names and data types of the columns in a table.
INDEXES FOR	are the keywords you use to display the names of each index, the owner of the index, the type (unique or not), and the names of the columns it covers.
ACCESS FOR	are the keywords you use to display the user access privileges for a table.

PRIVILEGES FOR	are synonymous keywords to ACCESS FOR.
STATUS FOR	are the keywords you use to display information on the owner, row length, number of columns, number of rows, the creation date, and the status of audit trails for a table.
table-name	is the name of the table you want information about.

Notes

1. You can also use the **Info** options on the RDSQL Menu or the TABLE Menu to display this information. Using the menu provides the same information as using the INFO statement within **RDSQL**.

2. For information about database-level privileges, you must use a SELECT statement to access the **sysusers** system catalog.

INSERT

Overview

Use the INSERT statement to insert one or more new rows into an existing table.

Syntax

```
INSERT INTO table-name [(column-list)]
        {VALUES (value-list) | SELECT-statement}
```

Explanation

INSERT INTO	are required keywords.
table-name	is the name of the table to which you want to add rows.
column-list	a list of the names of the columns into which you want to insert data. You can enter one column name or a series of column names, separated by commas.
VALUES	is a keyword.
value-list	are the values that you want to insert into the columns you specified. You can enter one value or a series of values, separated by commas. Character and date values must be enclosed in quotes.
SELECT-statement	is a valid **RDSQL** SELECT statement.

Notes

1. RDSQL inserts data into the columns in the specified table in the order in which you enter column names. It inserts the first value you enter into the first listed column, the second value into the second listed column, and so on.

2. Entering column names is optional. If you omit them, RDSQL assumes the values are listed in the order in which the columns are listed in the **syscolumns** systems catalog. Use the INFO COLUMNS FOR *table-name* statement to determine the column order. Unless you have subsequently used the ALTER TABLE statement to change the order, the order is the same as when the table was created.

3. RDSQL inserts the rows of data that result from the SELECT statement into the table, just as though you had entered them with the VALUES keyword. When using *value-list,* you may insert only one row at a time.

4. You cannot use *table-name* in the FROM clause of the SELECT statement. The data must be selected from other tables.

5. You may not include an INTO TEMP clause or an ORDER BY clause in the SELECT statement.

6. Although the values you insert do not have to be of the same data type as the columns themselves, they must be compatible. You can insert only CHAR data into CHAR columns and only numeric or character representation of numeric data into numeric columns.

7. Enter a zero (0) for a SERIAL column in the INSERT statement if you want RDSQL to insert the next SERIAL value for the table. Enter a nonzero value for a SERIAL column that does not duplicate a value already in the table, if you want RDSQL to use that value. If you enter a nonzero value for a SERIAL column that duplicates a value already in the table and if a unique index is defined on the column, it is an error.

8. All constant CHAR and DATE values must be enclosed in quotation marks.

9. When you create a database with transactions, each **RDSQL** statement that you execute is treated as a single transaction, even if you do not use the BEGIN WORK and COMMIT or ROLLBACK WORK statements. Because each row affected by the INSERT statement within a transaction is locked for the duration of the transaction, a single INSERT statement that affects a large number of rows locks those rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If this occurs, you may want either to reduce the scope of the INSERT statement or lock the entire table before executing the statement.

   See the "Locking" section in this chapter for a more detailed description of table-level and row-level locking in **INFORMIX-SQL**.

Caution! **RDSQL** makes every possible effort to perform data conversion, including converting the character string "123" into the integer 123. If the data cannot be converted, however, INSERT will stop. Unless you have created the database with transactions, all changes made to that point will remain, but subsequent rows from the SELECT statement will not be inserted. Data conversion will also fail if the target data type cannot hold the value offered. For example, you cannot insert the integer 123456 into a SMALLINT.

Examples

```
INSERT INTO customer
    VALUES (0, "Mary Anne", "Ronaldo", "Ski & Stuff",
        "21 Fig Road", "Grand-Lake Plaza", "Oakland",
            "CA", "94610", "415-555-4100");
```

```
create table newtable
        (item_num       smallint,
         order_num      integer,
         quantity       smallint,
         stock_num      smallint,
         manu_code      char(4),
         total_price    money(8)
        );
insert into newtable
       select item_num, order_num,
           quantity, stock_num, manu_code,
           total_price from items;
drop table items;
rename table newtable to items;
```

Related Statements

DELETE, SELECT

LOAD

Overview

Use the LOAD statement to fill an existing table with ASCII data taken from an operating system file.

Syntax

LOAD FROM *filename* INSERT INTO *table-name* [(*column-name*[, ...])]

Explanation

LOAD FROM	are required keywords.
filename	is the name of an operating system file that contains data you want to load. *filename* must be within quotation marks.
INSERT INTO	are required keywords.
table-name	is the name of the existing table in which you want to store the data.
column-name	is an optional column name in *table-name*.

Notes

1. Each column of the system file must be delimited by the character stored in the DBDELIMITER environment variable. The default delimiter is the vertical bar (| = ASCII 124).

2. Other than a NEWLINE character at the end of each record, the order and data type of the data fields in the operating system file must be identical to the order and data type of the columns specified in the database table.

The length of the data field must be less than or equal to the length specified in the database table.

3. An optional list of one or more column names, enclosed within parentheses and separated by commas, may be provided.

4. CHAR data fields exceeding the length of a column are truncated.

5. The number of data fields in the load file must equal the number of columns in the table (or in the optional list of column names).

6. A NULL column should be represented in the input file by no characters between delimiters, and a blank CHAR column, by one or more blanks. It is permissible to have leading blanks in numeric, DATE, and MONEY fields. DATE fields should have the format *mm/dd/yyyy*—where *mm* is the month (1 or 01 for January, and so on), *dd* is the day, and *yyyy* is the year. If the format of the DATE field is *mm/dd/yy*, the *yy* is interpreted as *19yy*. The DATE field must be a legal date (for example, February 30 is illegal). MONEY fields may have leading dollar signs.

7. You must have INSERT permission to use the LOAD statement.

8. If your database has transactions, you must execute the LOAD statement inside a transaction.

9. LOAD appends to rather than overwrites existing data.

10. See also the **dbload** utility described in Appendix J.

Examples

```
load from "/usr/enid/sample"
  insert into customer;

load from "/usr/alex/datafile"
  insert into customer
  (fname, lname, address1, city, state);
```

Related Statements

UNLOAD

LOCK TABLE

Overview

Use the LOCK TABLE statement to prohibit access to a table by other users.

Syntax

LOCK TABLE *table-name* IN {SHARE | EXCLUSIVE} MODE

Explanation

LOCK TABLE	are required keywords.
table-name	is the name of the table you want to lock.
IN	is a required keyword.
SHARE	is the keyword you use to give other users read access to the table, but to prevent them from modifying any of the data it contains.
EXCLUSIVE	is the keyword you use to prevent other users from having *any* access to the table.
MODE	is a required keyword.

Notes

1. Only one lock can apply to a table at any given time. That is, if a user locks a table (in either SHARE or EXCLUSIVE mode), no other user can lock that table in either mode until the first user unlocks it.

2. You can use the LOCK TABLE statement immediately after a BEGIN WORK statement to override row-level locking during the transaction. Normally, each row of a table affected by a statement within a transaction is locked for the duration of the transaction.

   During transactions that affect a large number of rows, you may exceed the limit placed by your operating system on the maximum number of locks. If you lock the entire table at the beginning of the transaction, however, **INFORMIX-SQL** does not lock each row in the table. You may want to use this strategy when executing a transaction that affects a large number of rows or every row in a table. The LOCK TABLE statement must be issued after the BEGIN WORK statement.

   See the "Locking" section in this chapter for further explanation of table-level and row-level locking in **INFORMIX-SQL**.

3. On a database with transactions, the COMMIT WORK statement removes all locks even if they were created using the LOCK TABLE statement.

4. In a database without transactions, the table remains locked until the user exits **INFORMIX-SQL** or executes the UNLOCK TABLE statement.

Examples

```
lock table orders in exclusive mode;
```

Related Statements

UNLOCK TABLE

OUTPUT

Overview

Use the OUTPUT statement to send query results directly to an operating system file or through a pipe to another program.

Syntax

OUTPUT TO {*filename* | PIPE *program*}
[WITHOUT HEADINGS] *SELECT-statement*

Explanation

OUTPUT TO	are required keywords.
filename	is the name of the operating system file in which you want to store the results of the query.
PIPE	is the keyword you use to send the query results to another program.
program	is the name of the program you want the query results sent to.
WITHOUT HEADINGS	are keywords telling **RDSQL** to display the results of a query without column name headings.
SELECT-statement	is the mandatory SELECT statement that selects the data you want to send to *filename* or *program*.

Examples

An example on a UNIX system:

```
output to /usr/gary/query1
   select * from customer;
```

An example on a DOS system:

```
output to \GARY\QUERY1
   select * from customer;
```

Related Statements

UNLOAD

RECOVER TABLE

Overview

In the event of a system failure, use the RECOVER TABLE statement to restore a database table from a backup copy and an audit trail file.

Syntax

RECOVER TABLE *table-name*

Explanation

RECOVER TABLE	are required keywords.
table-name	is the name of the table you want to recover.

Notes

1. Once you have recovered the table, use the DROP AUDIT statement to remove the contents of the audit trail file. Run the CREATE AUDIT statement to start a new audit trail file, then back up the table. See the "Audit Trails" section in this chapter for more information.

2. RECOVER TABLE checks that the audit trail and *table-name* have consistent record numbers for rows where changes have taken place. If RECOVER TABLE finds inconsistencies, it stops restoring the table.

3. You must own *table-name* or have DBA status to use the RECOVER TABLE statement.

Examples

The following list of RDSQL statements gives a template for the recovery of a table. They assume that your audit trail began from the last backup.

```
{restore table from last backup}
recover table customer;
drop audit for customer;
create audit for customer in "/dev/safe";
    select * from customer;
```

Related Statements

CREATE AUDIT, DROP AUDIT

RENAME COLUMN

Overview

Use the RENAME COLUMN statement to change the name of a column.

Syntax

RENAME COLUMN *table.oldcolumn* TO *newcolumn*

Explanation

RENAME COLUMN	are required keywords.
table	is the name of the table containing the column whose name you want to change. This is a required item in the statement.
oldcolumn	is the name of the column you want to rename.
TO	is a required keyword.
newcolumn	is the new name you want to assign to the column. *newcolumn* must satisfy the requirements for an **RDSQL** identifier and cannot duplicate another column name in the table.

Notes

1. You may RENAME a column of a table only when you own the table, have DBA privilege, or have been granted ALTER permission.

2. Do not use the RENAME COLUMN statement in a transaction; it cannot be rolled back.

Examples

```
rename column customer.customer_num to c_num;
```

Related Statements

ALTER TABLE, CREATE TABLE, INSERT, DROP TABLE, RENAME TABLE

RENAME TABLE

Overview

Use the RENAME TABLE statement to change the name of a table.

Syntax

RENAME TABLE *oldname* TO *newname*

Explanation

RENAME TABLE	are required keywords.
oldname	is the name of the table you want to rename.
TO	is a required keyword.
newname	is the new name you want to assign to the table.

Notes

1. You may RENAME a table only when you own the table, have DBA privilege, or have been granted ALTER permission on the table.

2. Do not execute the RENAME TABLE statement within a transaction; it cannot be rolled back.

Examples

This example moves the **quantity** column to the third place.

```
create table newtable
        (item_num       smallint,
         order_num      integer,
         quantity       smallint,
         stock_num      smallint,
         manu_code      char(3),
         total_price    money(8)
        );
insert into newtable
       select item_num, order_num,
           quantity, stock_num, manu_code,
           total_price from items;
drop table items;
rename table newtable to items;
```

Related Statements

ALTER TABLE, CREATE TABLE, INSERT, DROP TABLE, RENAME COLUMN

REPAIR TABLE

Overview

Use the REPAIR TABLE statement to delete and rebuild damaged indexes for a table. First, use the CHECK TABLE statement to determine if the data or the indexes are corrupted because of a power failure, computer crash, or other unintended program stoppage while you are using INFORMIX-SQL. REPAIR TABLE uses the **bcheck** utility. See Appendix G for a full description of **bcheck**.

Syntax

REPAIR TABLE *table-name*

Explanation

REPAIR TABLE	are required keywords.
table-name	is the name of the table whose indexes you want to repair.

Notes

1. You must own *table-name* or have the DBA privilege to use REPAIR TABLE.

2. You cannot use REPAIR TABLE on the system catalog **systables**.

Examples

```
repair table customer;
```

Related Statements

CHECK TABLE

REVOKE

Overview

Use the REVOKE statement to remove another user's access privileges for a table or database. Access privileges are relevant only on multi-user systems.

Syntax

REVOKE {*tab-privilege* ON *table-name* | *db-privilege*}
FROM {PUBLIC | *user-list*}

Explanation

REVOKE — is a required keyword.

tab-privilege — is one or more of the following table-level access privileges, separated by commas:

ALTER	Adds or deletes columns or modifies data types of columns
DELETE	Deletes rows
INDEX	Creates indexes
INSERT	Inserts rows
SELECT	Retrieves data
UPDATE	Changes column values
ALL [PRIVILEGES]	All of the above

ON — is a required keyword.

table-name is the name of the table for which you are revoking access privileges.

db-privilege is one of the following database-level access types:

CONNECT	allows access to database tables without permission to create permanent tables and indexes.
RESOURCE	allows access to database tables with permission to create permanent tables and indexes.
DBA	allows full database administrator privileges.

FROM is a required keyword.

PUBLIC is the keyword you use to revoke access *privilege* from all users.

user-list is a list of login names for the users whose access privilege you are revoking. You can enter one login name or a series of login names, separated by commas.

Notes

1. Do not execute the REVOKE statement within a transaction; it cannot be rolled back.

2. You can revoke database-level access privileges only if you have DBA status.

3. You can revoke only table-level access privileges that you have granted to another user.

4. You cannot revoke privileges from yourself.

5. Although you can grant UPDATE and SELECT privileges for specific columns, you cannot revoke these privileges column by column. If you revoke UPDATE or SELECT privileges from a user, **RDSQL** automatically revokes all UPDATE and SELECT privileges you have ever granted to that user for *table-name*. You can then re-grant privileges for specific columns.

6. Only a DBA recipient can revoke the DBA privilege from another recipient. If the database creator grants DBA privileges to another user, that person can revoke the DBA privilege from the database creator.

7. If you revoke the DBA or RESOURCE privilege from one or more users, they are left with the CONNECT privilege. To revoke all database privileges from users with DBA or RESOURCE status, you must revoke CONNECT as well as DBA or RESOURCE.

Examples

```
revoke delete, update
    on customer from john, mary;
```

Related Statements

GRANT

ROLLBACK WORK

Overview

Use the ROLLBACK WORK statement to undo all database modifications since the last BEGIN WORK statement.

Syntax

```
ROLLBACK WORK
```

Explanation

ROLLBACK WORK are required keywords.

Notes

1. ROLLBACK WORK releases all row and table locks.

2. See the "Transactions" section earlier in this chapter for further details.

Related Statements

BEGIN WORK, COMMIT WORK

ROLLFORWARD DATABASE

Overview

Use the ROLLFORWARD DATABASE statement to cause RDSQL to apply the transactions in the transaction log file to a backup copy of your database, recovering all completed transactions.

Syntax

ROLLFORWARD DATABASE *database-name*

Explanation

ROLLFORWARD DATABASE	are required keywords.
database-name	is the name of a database.

Notes

1. See the section "Transactions," earlier in this chapter, for more information.

Related Statements

BEGIN WORK, COMMIT WORK, START DATABASE, ROLLBACK WORK

SELECT

Overview

Use the SELECT statement to query the current database.

The SELECT statement can include up to seven clauses. Only the SELECT clause and the FROM clause are required.

Syntax

```
SELECT clause  FROM clause
               [WHERE clause]
               [GROUP BY clause]
               [HAVING clause]
               [ORDER BY clause]
               [INTO TEMP clause]
```

See "The SELECT Statement" section, later in this chapter, for detailed descriptions of these clauses.

SET LOCK MODE

Overview

Use the SET LOCK MODE statement to determine whether subsequent RDSQL calls wait for a locked row to become unlocked.

Syntax

```
SET LOCK MODE TO [NOT] WAIT
```

Explanation

SET LOCK MODE	are required keywords.
TO	is a required keyword.
NOT	is an optional keyword.
WAIT	is a required keyword.

Notes

1. The TO NOT WAIT option causes RDSQL to return an error if a statement attempts to alter or delete a row that another process has locked. This is the default situation; that is, if you have not issued a SET LOCK MODE statement previously. The NOT option is relevant, therefore, only when you have previously executed SET LOCK MODE TO WAIT and want to return to the default state.

2. The TO WAIT option causes RDSQL to wait on an attempt to alter or delete a row (or to SELECT a row FOR UPDATE) that another process has locked until the locked row becomes unlocked.

3. You should use the SET LOCK MODE TO WAIT statement with extreme caution. If the locking process fails and does not remove the lock, your statement can wait indefinitely.

4. This feature is available only on systems that have record-level locking and applies only to row-level locking. *Any* attempt by another user to access a row in a table locked IN EXCLUSIVE MODE produces errors.

START DATABASE

Overview

Use the START DATABASE statement to start a new transaction log file.

Syntax

START DATABASE *database-name* WITH LOG IN "*file-name*"

Explanation

START DATABASE	are required keywords.
database-name	is the name of a database.
WITH LOG IN	are required keywords.
file-name	is the full name of the transaction log file. *file-name* must be enclosed in quotation marks.

Notes

1. Use the START DATABASE statement only if you change the name of your transaction log file or if you want to start recording transactions in a database that was created without transactions.

2. See the section "Transactions" earlier in this chapter, for full details.

3. You must close the current database before issuing a START DATABASE statement.

4. The *database-name* then becomes the current database.

Related Statements

BEGIN WORK, COMMIT WORK, CREATE DATABASE, ROLLBACK WORK, ROLLFORWARD DATABASE

UNLOAD

Overview

Use the UNLOAD statement to write the data from a table in ASCII format to an operating system file.

Syntax

UNLOAD TO *pathname SELECT-statement*

Explanation

UNLOAD TO	are required keywords.
pathname	is the pathname of the operating system file in which you want to store the unloaded table.
SELECT-statement	is an **RDSQL** SELECT statement that retrieves the data you want to write to a file.

Notes

1. The data from each column in each row is separated from the data in the next column by the delimiter stored in the DBDELIMITER environment variable. The delimiter character should not appear otherwise in your data. The default delimiter is the vertical bar (| = ASCII 124).

2. CHAR data will have trailing blanks clipped, and NULL columns will be represented as no characters between delimiters. Numeric data will have no leading blanks. An INTEGER or SMALLINT zero will be represented as *0*; FLOAT, SMALLFLOAT, DECIMAL, and MONEY zeros will be represented as *0.00*. DATEs will be represented as

mm/dd/yyyy, where *mm* is the month (January = 1, and so on), *dd* is the day, and *yyyy* is the year. MONEY data will not have a leading dollar sign.

3. You must have SELECT permission on all columns in the *select-list* of the SELECT statement.

Examples

```
unload to "lname.out"
    select lname from customer;
```

Related Statements

LOAD, OUTPUT, SELECT

UNLOCK TABLE

Overview

Use the UNLOCK TABLE statement to unlock a table that you previously locked with the LOCK TABLE statement.

Syntax

UNLOCK TABLE *table-name*

Explanation

UNLOCK TABLE	are required keywords.
table-name	is the name of the table you want to unlock.

Notes

1. If the database has transactions, you cannot use the UNLOCK TABLE statement. All row or table locks are released when a COMMIT WORK or ROLLBACK WORK statement is processed.

Related Statements

LOCK TABLE

UPDATE

Overview

Use the UPDATE statement to change the values in one or more columns of one or more rows in a table.

Syntax

```
UPDATE table-name SET {column-name = expr[, ... ] |
        {(column-list) | *} = (expr-list)}
  [WHERE condition]
```

Explanation

UPDATE	is a required keyword.
table-name	is the name of the table that contains the row(s) you want to update.
SET	is a required keyword.
column-name	is the name of a column you want to update.
expr	is any combination of column names, constants, or arithmetic operators, or an **RDSQL** subquery that returns a single row of one value.
column-list	is a list of the names of columns you want to update.
*	refers to all columns in *table-name*.

expr-list is a list of expressions that represent values corresponding to the columns in *column-list* or the columns represented by the asterisk notation. The list can include an **RDSQL** subquery that returns a single row of multiple values.

WHERE is an optional keyword.

condition is a condition for a standard WHERE clause made up of a search condition that compares the values in one column to the values in another column, or to a constant. (For further information, refer to the explanation of WHERE clauses in the section "The SELECT Statement" at the end of this chapter.)

Notes

1. *expr* may be a SELECT statement in parentheses that adheres to standard rules for subqueries. The SELECT statement can return no more than one value except when included in an *expr-list*.

2. You cannot use a SELECT statement that retrieves data from *table-name*.

3. The number of column names included in the *column-list* must equal the number of values produced in the *expr-list*.

4. Although the value returned by *expr* does not have to be of the same data type as *column-name*, it must be compatible. You can put only CHAR data into CHAR columns and only numeric or character representations of numeric data into numeric columns.

5. For databases with transactions, all **RDSQL** statements executed within a BEGIN WORK and COMMIT WORK/ROLLBACK WORK block are treated as a single transaction. Because each row affected by an UPDATE statement within a transaction is locked for the duration of the transaction, a single UPDATE statement that affects a

large number of rows will lock those rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If this occurs, you may want either to reduce the scope of the UPDATE statement or lock the entire table before executing the statement. See the section in Chapter 2 entitled "Locking" for a more detailed description of table- and row-level locking in **RDSQL**.

6. You cannot use the UNIQUE keyword in a subquery within an UPDATE statement.

Caution! If RDSQL encounters an error while performing an UPDATE, the operation stops. Unless you have created the database with transactions, all database changes made up to the point where the error is encountered remain, but subsequent rows are not updated.

A data conversion error is an example of an error that stops an UPDATE operation. Common data conversion errors include attempting to insert numeric data into a CHAR column, or attempting to insert numeric values that exceed the limits of the data type of the column. For example, you cannot insert the integer 123456 into a SMALLINT column.

If you omit the WHERE clause, RDSQL assumes you want to update every row in the table.

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 command line, RDSQL executes it immediately.

See Appendix M for information about running RDSQL from the command line.

Examples

```
update stock
   set unit_price = unit_price * 1.04
   where manu_code = "HRO";


update customer
   set (fname, company, address2) =
         ("Marie", "Marie's Sports", "P. O. Box 3621")
   where customer_num = 103;


update table1
   set (col1, col2, col3) =
         ((select min (ship_charge), max (ship_charge)
         from orders), "07/01/1986")
   where col4 = 1001;
```

Related Statements

SELECT, DELETE, INSERT

UPDATE STATISTICS

Overview

Use the UPDATE STATISTICS statement to cause the number of rows in a table to be recorded in the **systables** catalog.

Syntax

UPDATE STATISTICS [FOR TABLE *table-name*]

Explanation

UPDATE STATISTICS	are required keywords.
FOR TABLE	are optional keywords you use when you want to update the statistics for a single table.
table-name	is the name of the table for which you want the statistics updated.

Notes

1. UPDATE STATISTICS is effective only when there is a current database.

2. **RDSQL** uses the data generated by UPDATE STATISTICS to optimize searching strategy. When you have modified a table extensively, use UPDATE STATISTICS to improve the efficiency of queries.

3. **RDSQL** does not update the statistics unless you execute the UPDATE STATISTICS statement.

4. If you do not use the FOR TABLE clause, UPDATE STATISTICS will update all the tables in the current database.

The SELECT Statement

Overview

Use the SELECT statement to query the current database.

The SELECT statement is made up of the following seven clauses. Only the SELECT clause and the FROM clause are required.

Syntax

```
SELECT clause  FROM clause
               [WHERE clause]
               [GROUP BY clause]
               [HAVING clause]
               [ORDER BY clause]
               [INTO TEMP clause]
```

These have the following syntax:

```
SELECT [ALL | DISTINCT | UNIQUE] select-list
```

```
FROM [OUTER] table-name [table-alias] [,...]
```

```
WHERE condition
```

A *condition* is a collection of one or more *search conditions* connected by the logical operators AND, OR, or NOT. A search condition can be any of the following:

1. Comparison Condition

   a. *expr rel-op expr*

   b. *expr* [NOT] BETWEEN *expr* AND *expr*

   c. *expr* [NOT] IN (*value-list*)

   d. *column-name* [NOT] LIKE "*string*"

   e. *column-name* [NOT] MATCHES "*string*"

   f. *column-name* IS [NOT] NULL

2. Join Condition (a comparison condition among columns of the joined tables)

3. Condition with Subquery

   a. *expr rel-op* {ALL | ANY | SOME} (*SELECT-statement*)

   b. *expr* [NOT] IN (*SELECT-statement*)

   c. [NOT] EXISTS (*SELECT-statement*)

GROUP BY *column-list*

HAVING *condition*

ORDER BY *column-name* [ASC | DESC][, ...]

Explanation

The following pages explain each of the syntax elements. A few basic concepts are defined here.

1. An *expression* consists of a column name, or a constant, or any combination of these connected by the arithmetic operators:

Operator	Operation
+	addition
−	subtraction
*	multiplication
/	division

The result of the operation must make sense. For example, you cannot divide 16 by Jones.

RDSQL has two functions that can be used wherever a constant can be used. TODAY always returns your system's date. USER returns a string containing the current user's name. On UNIX systems, this is the login name, while on multi-user DOS systems, this is the machine name.

An expression may also be one of the aggregate or one of the date functions. You may not include both an aggregate function and a column in an expression. The aggregate and date functions that you can use in **RDSQL** statements are defined at the end of this chapter.

A CHAR column may have subscripts so that only a portion of the column value is involved in the expression. The notation for subscripting a column is *column-name*[*m*, *n*], where you want the *m*th through the *n*th characters of *column-name*. *m* must be less than or equal to *n*.

2. A *relational operator* is one of the following:

Operator	Operation
=	equal
!= or < >	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

For CHAR expressions, greater than means "after" in ASCII collating order, where lowercase letters are after uppercase letters, and both are after numerals. For the ASCII collating sequence of all the characters, see the ASCII chart in Appendix F.

For DATE expressions, greater than means later in time.

Notes

1. The clauses of the SELECT statement are explained in detail on the following pages. Briefly, the SELECT clause names a list of columns or expressions to be retrieved, the FROM clause names a list of tables, the WHERE clause sets conditions on the rows, the GROUP BY clause groups rows together, the HAVING clause sets conditions on the groups, the ORDER BY clause orders the selected rows, and the INTO TEMP clause puts the results into a temporary table.

2. It is sometimes helpful to think of the SELECT statement as follows:

   When you list more than one table in the FROM clause, **RDSQL** behaves as though it were creating a composite table that is the Cartesian product of all the tables in the FROM clause. That is, the rows of the new table are constructed by taking all the possible combinations of rows from all the tables listed in the FROM clause. If there is a WHERE clause, **RDSQL** eliminates from this new table all rows that do not meet the conditions of the WHERE clause. This modified table is returned to the SELECT clause, where all columns not listed are eliminated. The resulting table is what the SELECT statement returns.

SELECT Clause

Overview

Use the SELECT clause to specify the data you want to retrieve from one or more tables in a database.

Syntax

SELECT [ALL | DISTINCT | UNIQUE] *select-list*

Explanation

SELECT	is a required keyword.
ALL	is a keyword that causes **RDSQL** to select all rows that satisfy the WHERE clause, without eliminating duplicates. This keyword is the default.
DISTINCT	is a keyword that causes **RDSQL** to eliminate duplicate rows from the query results.
UNIQUE	is a keyword that is synonymous with DISTINCT.
select-list	is a list of column names and/or expressions separated by commas. A column name must be unambiguous; use its table name as a prefix if there can be confusion.

Notes

1. If the SELECT statement does not include a WHERE clause, every row will be returned.

2. You can use DISTINCT or UNIQUE only once in a query.

3. You may use the asterisk (*) in the *select-list* to select all columns from all the tables in the FROM clause. You could produce the same result by listing every column name in the *select-list.*

4. To select all the columns from a single table, use the notation *tablename.**.

5. You can supply a *display label* for the column name or an expression in the *select-list* by following the column name or expression with a legal identifier. The display label appears as the heading for that column or expression in the output of the SELECT statement. If you create a temporary table with the INTO TEMP clause, the column names of the temporary table are the display labels if they have been defined.

6. If you specify an aggregate function and a column in the *select-list*, the column must be used in the GROUP BY list (see GROUP BY for further explanation).

Examples

The following examples will use FROM and WHERE clauses, whose syntax will be defined later.

```
select customer_num, lname, city
    from customer;
```

This example selects columns **customer_num**, **lname**, and **city**; the FROM clause indicates that these columns are taken from the **customer** table.

```
select count(*)
    from orders
    where customer_num = 101;
```

This statement counts the number of rows in **orders** in which the **customer_num** column contains the value 101. In other words, it counts the number of orders made by the customer whose identifying number is 101.

```
select avg(total_price)
    from items
    where order_num = 1008;
```

This statement computes the average of the **total_price** values in those rows of **items** that contain an **order_num** column equal to 1008.

```
select a+b abtotal, c*d cdprod
    from tablez
    into temp x;
```

This example illustrates the use of display labels. It selects the sum of columns **a** and **b** from **tablez** and gives the sum the label **abtotal**. Similarly, the product of columns **c** and **d** is labeled **cdprod**.

FROM Clause

Overview

Use the FROM clause to specify the table or tables you want to select data from.

Syntax

```
FROM {table-name [table-alias] |
    OUTER table-name [table-alias] |
    OUTER (table-expr)} [,...]
```

Explanation

FROM	is a required keyword.
OUTER	is an optional keyword.
table-name	is the name of a table that contains data for which you are searching.
table-alias	is an optional alias for *table-name*.
table-expr	is one or more of the options in the syntax shown above, for example ..., tab1, outer tab2 ... (See Appendix L, "Complex Outer Joins," for a discussion of this syntax.)

Notes

1. Use the optional keyword OUTER to form outer joins. See the "Outer Joins" section in this chapter and Appendix L for a detailed description of outer joins.

2. You can supply an alias for a table name by following the table name with a space and an RDSQL identifier. This

feature is especially useful when performing self-joins (see the "WHERE Clause" section of this chapter). Once an alias is supplied, you must use the alias whenever referring to the table.

Examples

The following example selects customers who have placed orders.

```
select fname, lname, order_num
   from customer, orders
   where customer.customer_num = orders.customer_num
```

The following example selects all customers whether or not they have placed orders.

```
select fname, lname, order_num
   from customer, outer orders
   where customer.customer_num = orders.customer_num
```

Overview

Use the WHERE clause to specify search criteria and join conditions on the data you want to select.

Syntax

WHERE *condition*

Explanation

WHERE is a required keyword.

condition is a collection of one or more *search conditions* connected by the logical operators AND, OR, or NOT. A search condition can be any of the following:

- Comparison Condition
- Join Condition
- Condition with Subquery

Comparison Condition

A comparison condition can have one of the following forms:

a. *expr rel-op expr*

b. *expr* [NOT] BETWEEN *expr* AND *expr*

c. *expr* [NOT] IN (*value-list*)

d. *column-name* [NOT] LIKE "*string*"

e. *column-name* [NOT] MATCHES "*string*"

f. *column-name* IS [NOT] NULL

These forms are explained in the following pages. Any one of these conditions can be preceded by the keyword NOT; in which case, the negative of the condition must be satisfied.

Syntax

---

expr rel-op expr

Explanation

expr is an expression.

rel-op is a relational operator.

Examples

```
select customer_num, order_date
    from orders
    where paid_date is null;

select fname, lname, company
    from customer
    where city[1,3] = "San";

select order_num, company
    from orders o, customer c
    where o.order_date > "6/04/86"
        and o.customer_num = c.customer_num;
```

Syntax

expr [NOT] BETWEEN *expr* AND *expr*

Explanation

expr	is an expression.
NOT	is a keyword that indicates that the expression to the left lies outside the range.
BETWEEN	is a keyword that indicates that the value of the expression to its left lies in the inclusive range of the values of the two expressions to its right.
AND	is a required keyword.

Examples

```
select stock_num, manu_code
    from stock
    where unit_price between
        125.00 and 200.00;

select unique customer_num, stock_num, manu_code
    from orders, items
    where order_date
        between "6/1/86" and "6/7/86"
        and orders.order_num = items.order_num;

select fname, lname
    from customer
    where zipcode not
        between "94100" and "94199";
```

Syntax

expr [NOT] IN (*value-list*)

Explanation

expr is an expression.

NOT is an optional keyword.

IN is a required keyword.

value-list is a list of values enclosed in parentheses.

Notes

1. The search condition is satisfied when the expression to the left is included in the list of items. The NOT option produces a search condition that is satisfied when *expr* is not in the list of items.

2. *value-list* may contain constants or the special keyword constants TODAY and USER , which contain the current date and the user's login.

Examples

```
select lname, fname, company
    from customer
    where state in ("CA", "WA", "OR");

select item_num, total_price
    from items
    where manu_code in
        ("HRO", "HSK");
```

Syntax

column-name [NOT] LIKE "*string*"

Explanation

column-name	is the name of a column.
NOT	is an optional keyword.
LIKE	is a required keyword.
string	is a pattern of characters enclosed in quotation marks.

Notes

1. The search condition is successful when the value of the column on the left matches the pattern specified by *string*. You can use wildcard characters in place of other characters in the string:

   % A percent sign matches zero or more characters.

   _ An underscore matches any single character.

2. The NOT option makes the search condition successful when the column on the left does not match the pattern specified by *string*.

Examples

```
select fname, lname
    from customer
    where lname like "%son";
```

finds every customer representative whose last name ends in *son*.

```
select stock_num, manu_code, unit_price
    from stock
    where description like "%ball%";
```

obtains stock number, manufacturer's code, and unit price for all stock items with *ball* anywhere in their description.

Syntax

column-name [NOT] MATCHES "*string*"

Explanation

column-name	is the name of a column.
NOT	is an optional keyword.
MATCHES	is a required keyword.
string	is a pattern of characters enclosed in quotation marks.

Notes

1. The search condition is successful when the value of the column on the left matches the pattern specified by *string*. You can use three wildcard characters in place of other characters in the string:

*	matches zero or more characters
?	matches any single character
[...]	matches any one of the enclosed characters or character ranges as in [a-z]. [acf] means **a,** or **c,** or **f.** [a-f] means **a,** or **b,** or **c,** or **d,** or **e,** or **f.** A caret (^) as the first character within the brackets matches any character that is *not* listed. [^abc] matches any character that is not **a** or **b** or **c**.
\	escapes special significance of the next character (used to match * or ? by writing * or \?)

2. The NOT option makes the search condition successful when the column on the left does not match the pattern specified by *string*.

3. The MATCHES comparison is an RDS extension to retain compatibility with earlier versions of INFORMIX.

4. Values used in a MATCHES search must be type CHAR.

Examples

```
select fname, lname
    from customer
    where lname matches "Richard*";
```

selects rows in which the first seven letters of the last name are *Richard* (thus matching Richard, Richards, Richardson, and any others).

```
select customer_num, company
    from customer
    where city matches "[A-J]*";
```

provides the customer number and company name for all customers in cities that start with the letters *A* through *J*.

Syntax

column-name IS [NOT] NULL

Explanation

column-name	is the name of a column.
IS NULL	are required keywords.
NOT	is an optional keyword.

Examples

```
select order_num, customer_num
    from orders
    where paid_date is null;
```

lists those order numbers and customer numbers where the order has not been paid.

You can link any number of the above described conditions together using the logical operators AND or OR. For example:

```
select order_num, total_price
    from items
    where total_price > 1000.00 and
        manu_code like "H%";

select lname, customer_num
    from customer
    where zipcode between
        "93500" and "95700"
        or state not in
        ("CA", "WA", "OR");
```

Join Conditions

Overview

You join two tables when you create a relationship in the WHERE clause between at least one column from one table and at least one column from the other. The effect of the join is to create a temporary composite table in which each pair of rows (one from each table) satisfying the join condition is linked together to form a single row.

Syntax

The critical elements of a SELECT statement with a join between two tables *table1* and *table2* follow:

SELECT clause FROM *table1*, *table2* WHERE *condition*

Explanation

SELECT clause	is any valid SELECT clause involving columns from *table1* or *table2*.
FROM	is a required keyword.
table1	is one of the tables in the join.
table2	is the other table in the join.
WHERE	is a required keyword.
condition	is any of the comparison conditions that involves columns from both *table1* and *table2*.

Notes

1. When columns from different tables have the same name, you must distinguish them by prefixing the table identifier and a period—**table.column**.

2. A *multiple join* is a join of more than two tables. Its structure is similar to that shown previously, except that you have a join condition for more than one pair of tables in the FROM clause.

3. You can also join a table to itself in a *self-join*. To do so, you must list the table name twice in the FROM clause, assigning it two different aliases. Use the aliases to refer to each of the "two" tables in the WHERE clause.

4. An *outer join* occurs when every row from *table1* is taken whether or not the join condition is met. If the join condition is not met, the columns from *table2* in the *select-list* are set to NULL values. You indicate an outer join by inserting the keyword OUTER before the table name *table2*.

Examples

A two-table join:

```
select order_num, lname, fname
    from customer, orders
    where customer.customer_num
        = orders.customer_num;
```

lists the order number and first and last names of the customer's representative for each order.

A multiple-table join:

```
select unique company, stock_num, manu_code
    from customer c, orders o, items i
    where c.customer_num = o.customer_num
        and o.order_num = i.order_num;
```

yields the company name of the customer who ordered an item identified with the stock number and manufacturer code.

Self-join:

```
select x.stock_num, x.manu_code,
        y.stock_num, y.manu_code
    from stock x, stock y
    where x.unit_price > 2.5 * y.unit_price,
```

finds pairs of stock items whose unit prices differ by a factor greater than two and one-half. **x** and **y** are each aliases for the **stock** table.

Outer Join:

```
select company, order_num
    from customer c,
    outer orders o
    where c.customer_num = o.customer_num;
```

will list all the customers' company name and order numbers, if the customer has placed an order. If not, the company name will still be listed. See the "Outer Joins" section in this chapter and Appendix L for information about outer joins.

Subqueries

Overview

The search condition in a SELECT statement can also

- Compare an expression to the result of another SELECT statement
- Determine whether an expression is included in the results of another SELECT statement
- Ask whether there are any rows selected by another SELECT statement

SELECT statements within a WHERE clause are called *subqueries*. A subquery may return a single value, no values, or a set of values, but it must have only a single column or expression in its *select-list* and must not contain an ORDER BY clause. The subquery can be dependent upon the current row being evaluated by the outer SELECT statement (correlated subqueries).

Syntax

WHERE *expr rel-op* {ALL | [ANY | SOME]} (*SELECT-statement*)

WHERE *expr* [NOT] IN (*SELECT-statement*)

WHERE [NOT] EXISTS (*SELECT-statement*)

Explanation

WHERE is a required keyword.

expr is an expression.

rel-op is a relational operator.

ALL is a keyword that denotes that the subquery may return zero, one, or more values and that the search condition is true if the comparison is true for each of the values returned. If the subquery returns no value, the search condition is true.

ANY is a keyword that denotes that the subquery may return zero, one, or more values and that the search condition is true if the comparison is true for at least one of the values returned. If the subquery returns no value, the search condition is false.

SOME is an alias for ANY.

IN is a keyword that asks whether *expr* is among the values returned by the following *SELECT-statement.*

EXISTS is a keyword that asks whether there are any rows returned by the following *SELECT-statement.* The search condition is true if the subquery returns one or more rows.

NOT is an optional keyword that reverses the truth value of the search condition.

Notes

1. The keywords ANY and ALL may be omitted in a comparison if you know the subquery will return exactly one value. In this case, the search condition is true if the comparison is true for the expression and the value returned by the subquery. It is an error if the subquery returns other than a single value.

2. *expr* IN (*SELECT-statement*) is equivalent to *expr* =ANY (*SELECT-statement*).

3. *expr* NOT IN (*SELECT-statement*) is equivalent to *expr* !=ALL (*SELECT-statement*).

Examples

```
select order_num
    from items
    where stock_num = 9 and quantity =
        (select max(quantity) from items
         where stock_num = 9);
```

This subquery returns a single value (the maximum number of volleyball nets ordered) to the outer-level query. The entire SELECT statement lists the order numbers for orders that include the maximum number of volleyball nets.

```
select unique order_num
    from items
    where total_price >all
        (select total_price
            from items
            where order_num = 1011);
```

This query lists the order numbers of all orders containing an item whose total price is greater than the total price on all the items in order number 1011.

```
select unique customer_num
    from orders
    where order_num not in
        (select order_num
            from items
          where stock_num = 1);
```

This query lists all customer numbers corresponding to customers who have placed orders that do not include baseball gloves (**stock_num** = 1).

```
select order_num, stock_num, manu_code, total_price
    from items x
    where total_price >
        (select 2 * min(total_price)
            from items
            where order_num = x.order_num);
```

This query (using a correlated subquery) lists all items whose total price is at least twice the price for the least expensive item on the same order.

GROUP BY Clause

Overview

Use the GROUP BY clause to produce a single row of results for each group. A group is a set of rows having the same values for each column listed.

Syntax

GROUP BY *group-list*

Explanation

GROUP BY	are required keywords.
group-list	is a column name or a list of column names, separated by commas, that determines the group. The query result contains a single row for each set of rows that satisfies the WHERE clause and contains a unique value or set of values in the column or columns indicated by *group-list*.

Notes

1. Using a GROUP BY clause restricts what you can enter in the SELECT clause. The *select-list* may include aggregate functions for any column and/or the name of any column that you also list in the GROUP BY clause. You may not, however, list any column in the *select-list* that you do not also list in *group-list*.

2. You cannot list a column in *group-list* that participates in *select-list* only as a part of an expression.

3. In the place of column names in *group-list*, you can enter one or more integers that refer to the position of items in the *select-list*.

Examples

```
select order_num, count(*), sum(total_price)
    from items
    group by order_num;
```

obtains the number of items and total price of all items for each order.

```
select order_num, count(*), sum(total_price)
    from items
    group by 1;
```

returns the same information as the previous example.

HAVING Clause

Overview

Use the HAVING clause to apply one or more qualifying conditions to groups.

Syntax

HAVING *condition*

Explanation

HAVING is a required keyword.

condition is a condition as defined for the WHERE clause.

Notes

1. Each condition compares one aggregate property of the group either with another aggregate property of the group or with a constant.

2. The HAVING clause generally complements a GROUP BY clause. If you use HAVING without GROUP BY, the HAVING clause applies to all rows that satisfy the WHERE clause. Without a GROUP BY clause, all rows that satisfy the WHERE clause make up a single group.

Examples

```
select order_num, avg(total_price)
    from items
    group by order_num
    having count(*) > 2;
```

This query returns average total price per item on all orders that have more than two items.

ORDER BY Clause

Overview

Use the ORDER BY clause to sort query results by the values contained in one or more columns. You can sort only by a column that you select in the SELECT clause.

Syntax

ORDER BY *column-name* [ASC | DESC][, ...]

Explanation

ORDER BY	are required keywords.
column-name	is the name of a column by which you want to sort the query results.
ASC	is a keyword that specifies that the results should be in ascending order. ASC is the default.
DESC	is a keyword that specifies that the results should be in descending order.

Notes

1. You can ORDER BY up to eight columns.
2. You can only ORDER BY columns that are named explicitly or implicitly in the *select-list.*
3. The total length of the data in the columns included in the ORDER BY clause cannot be greater than 120 bytes.

4. In the place of column names, you can enter one or more integers that refer to the position of items in the *select-list*. In this way, you can ORDER BY an expression.

Examples

```
   CORRECT:    select order_date, ship_date
                  from orders
                  order by order_date;

   CORRECT:    select *
                  from orders
                  order by order_date;

 INCORRECT:    select order_date, ship_date
                  from orders
                  order by customer_num;
```

In the first two examples, **order_date** is either explicitly or implicitly listed in the *select-list*. In the third example, even though **customer_num** is in the **orders** table, it was not contained in the *select-list*.

```
   select customer_num, fname, lname, company
      from customer
      order by 3, 2;
```

performs a nested sort, the first level being the **lname** column, the second level being the **fname** column.

INTO TEMP Clause

Overview

Use the INTO TEMP clause to create a temporary table that contains the query results. The temporary table disappears when you exit from the **INFORMIX-SQL** session.

Syntax

INTO TEMP *table-name*

Explanation

INTO TEMP	are required keywords.
table-name	is the **RDSQL** identifier you want to assign to the temporary table.

Notes

1. You will save time if you use a temporary table when the same query results are required several times.

2. An INTO TEMP clause in a SELECT statement often gives you clearer and more easily understood statements.

3. The column names of the temporary table are those named in the *select-list*. If you list a display label for a column or expression, the column name in the temporary table is the display label.

4. There are no indexes associated with *table-name*.

5. The temporary table persists until you end your **INFORMIX-SQL** session or issue the DROP TABLE statement for it.

UNION Operator

Overview

The UNION operator is a keyword placed between two SELECT statements that let you combine the queries into a single query.

Syntax

SELECT-statement UNION [ALL] *SELECT-statement*
[UNION [ALL] *SELECT-statement* ...]

Explanation

UNION	is a keyword that selects all rows from both queries, removes duplicates, and returns what is left.
ALL	is an optional keyword that leaves the duplicates.

Notes

1. The UNION operator can be placed between each member of a sequence of more than two queries.

2. It is possible to write single queries that are equivalent to the compound queries constructed using the UNION operator. The advantage of the UNION operator is that it makes the process easier.

3. There are restrictions on the queries that you may connect with UNION operators.

- The number of the items in the *select-list* of each query must be the same, and corresponding items in each *select-list* must have identical data types.

- Corresponding items need not have the same identifier.

- If you use an ORDER BY clause, it must follow the last SELECT statement, and you must refer by integer, not by identifier, to the item to be ordered. Ordering takes place after the set operation is complete.

4. UNION operators may not occur inside a subquery and may not be used in the definition of a view.

5. The column names (or display labels) of the resulting table are the same as those from the first SELECT statement.

6. You may put the results of a UNION into a temporary table by putting an INTO TEMP clause in the final SELECT statement.

Examples

The following example selects those items (identified by a stock number and a manufacturing code) that have a unit price of less than $100.00 or that have been ordered in quantities greater than 3.

```
select distinct stock_num, manu_code
    from stock
    where unit_price < 100.00

union

select stock_num, manu_code,
    from items
    where quantity > 3
    order by 1;
```

Aggregate and Date Functions

You may use the aggregate and the date functions anywhere in an **RDSQL** expression that a constant may appear. These functions are

Aggregate Functions	**Date Functions**
COUNT(*)	DATE( )
SUM( )	DAY( )
AVG( )	MDY( )
MAX( )	MONTH( )
MIN( )	WEEKDAY( )
	YEAR( )

They are defined on the following pages.

Aggregate Functions

Overview

The aggregate functions take on values that depend on the set of rows returned by the WHERE clause of a SELECT statement. In the absence of a WHERE clause, the aggregate functions take on values that depend on all the rows formed by the FROM clause.

Syntax	Function
COUNT(*)	returns the number of rows that satisfy the WHERE clause.
COUNT(DISTINCT *x*)	returns the number of unique values in column *x* that satisfy the WHERE clause.
SUM([DISTINCT] *x*)	returns the sum of all values in column *x* that satisfy the WHERE clause. You can apply SUM only to numeric columns. If you use the keyword DISTINCT, the sum is only over distinct values in column *x*.
AVG([DISTINCT] *x*)	returns the average of all values in column *x* that satisfy the WHERE clause. You can apply AVG only to numeric columns. If you use the keyword DISTINCT, the average is only over distinct values in column *x*.
MAX(*x*)	returns the highest value contained in column *x* from a row that satisfies the WHERE clause.

MIN(x)	returns the lowest value contained in column x from a row that satisfies the WHERE clause.

Notes

1. In the functions SUM(x), AVG(x), MAX(x), and MIN(x), x may be an expression instead of a column. In this case, the function is evaluated over the values of the expression as computed for each row that satisfies the WHERE clause. The expression may not contain another aggregate function.

2. You may use the DISTINCT keyword only with columns, not with expressions.

3. In a *select-list* you may use the keyword DISTINCT only once: either to eliminate duplicate query results or to eliminate duplicates from the argument of an aggregate function.

4. NULL values affect the aggregate functions as follows: COUNT(*) counts all rows, even if the value of every column in the row is NULL. COUNT(DISTINCT x), AVG (x), SUM (x), MAX (x), and MIN (x) ignore rows with NULL values for x and return the appropriate values based on the rest of the rows. If x contains only NULL values, however, then COUNT(DISTINCT x) returns zero, and the other four aggregate functions return NULL for that column.

DATE()

Overview

The DATE function returns a type DATE value corresponding to the expression you call it with.

Syntax

DATE(*expr*)

Explanation

DATE	is a required keyword.
expr	is a required expression that can be converted to a type DATE value.

Examples

The following example uses DATE to convert a string to a date.

```
where end_date > date("12/13/1984")
```

date(100) returns the 100^{th} day after December 31, 1899.

DAY()

Overview

The DAY function returns the day of the month when you call it with a type DATE expression.

Syntax

DAY(*date-expr*)

Explanation

DAY	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
select order_num, day (order_date) from orders;
```

MDY()

Overview

The MDY function returns a type DATE value when you call it with three expressions that evaluate to integers representing the month, date, and year.

Syntax

MDY(*expr1*, *expr2*, *expr3*)

Explanation

MDY	is a required keyword.
expr1	is an expression that evaluates to an integer representing the number of the month (1-12).
expr2	is an expression that evaluates to an integer representing the number of the day of the month (1-28, 29, 30, or 31, depending on the month).
expr3	is an expression that evaluates to a four-digit integer representing the year.

Notes

1. The value of *expr3* cannot be the abbreviation for the year. 86 is in the first century.

MONTH()

Overview

The MONTH function returns an integer corresponding to its type DATE argument.

Syntax

MONTH(*date-expr*)

Explanation

MONTH	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
select order_num, month (order_date) from orders;
```

WEEKDAY()

Overview

The WEEKDAY function returns an integer that represents the day of the week when you call it with a type DATE expression.

Syntax

WEEKDAY(*date-expr*)

Explanation

WEEKDAY	is a required keyword.
date-expr	is a required expression of type DATE.

Notes

1. WEEKDAY returns an integer in the range 0-6.
   0 represents Sunday, 1 represents Monday, and so on.

Examples

```
select order_num, weekday (order_date) from orders;
```

YEAR()

Overview

The YEAR function returns an integer that represents the year when you call it with a type DATE expression.

Syntax

YEAR(*date-expr*)

Explanation

YEAR	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
select order_num, year (order_date) from orders;
```

Chapter 3

The FORMBUILD Transaction Form Generator

Chapter 3 Table of Contents

Overview 5
PERFORM Error Messages 5
Demonstration Database Sample Forms 5

Creating and Compiling a Custom Form 6
Using Menus to Create a Form 6
Using the Operating System to Create a Form 8

Structure of a Form Specification File 10

DATABASE Section 12

SCREEN Section 13
Display Fields 15

TABLES Section 19

ATTRIBUTES Section 21
Display Field Order 21
Table Order 22
Fields Linked to Database Columns 23
Display-Only Fields 25
Joining Columns 28
Verify Joins 29
ATTRIBUTES Syntax 31
AUTONEXT 32
COMMENTS 34
DEFAULT 36
DOWNSHIFT 38
FORMAT 39
INCLUDE 42
LOOKUP 44
NOENTRY 46
NOUPDATE 48
PICTURE 50
QUERYCLEAR 52
REQUIRED 54
REVERSE 56

RIGHT 57
UPSHIFT 58
VERIFY 60
ZEROFILL 62

INSTRUCTIONS Section 63
COMPOSITES 64
DELIMITERS 66
MASTER OF 68
Control Blocks 71
BEFORE 72
AFTER 74
EDITADD and EDITUPDATE 76
ADD 79
UPDATE 81
QUERY 83
REMOVE 85
DISPLAY 87
Action Syntax 89
ABORT 90
LET 92
NEXTFIELD 95
COMMENTS 97
IF-THEN-ELSE 99

The *sample* Form Specification File 101
The CUSTOMER INFORMATION Screen 103
The ORDER INFORMATION Screen 103

Overview

Before you can use **PERFORM** with a customized screen form (see Chapter 4, "The PERFORM Screen Transaction Processor"), you must use **FORMBUILD** to compile a form specification file. The form specification file contains the screen format and the instructions to **PERFORM** about how to display the data.

PERFORM Error Messages

The text of all error messages, along with suggestions for corrections, is included in a special section at the back of this manual.

Demonstration Database Sample Forms

Examples in the *INFORMIX-SQL User Guide* and the *INFORMIX-SQL Reference Manual* are based on the following three sample form specifications. (One additional form has been included in the demonstration database to provide further practice.) These form specifications illustrate a wide variety of commands available with **PERFORM**.

customer.per	A simple form used to enter and retrieve customer information.
orderform.per	A more complex form used to enter and retrieve information about customer orders.
sample.per	The most sophisticated form included with the **stores** database. It is an expanded version of the **orderform** form.

Appendix E contains the full text of these sample form specifications. In addition, a copy of the **sample** specification is included at the end of this chapter.

Creating and Compiling a Custom Form

You can create a form specification file in one of two ways: You can use the **Form** option on the INFORMIX-SQL Main Menu, or work directly with the appropriate programs from the operating system command line. Either alternative requires that you have already created the database and all the tables to which the form will refer. The following two subsections describe these alternative procedures. They do not, however, describe the rules for constructing or modifying the form specification file. These rules are defined in the remaining sections of this chapter.

Using Menus to Create a Form

To create a customized screen form using the **INFORMIX-SQL** Menu system, follow these steps:

1. Select the **Form** option on the INFORMIX-SQL Main Menu, and then the **Generate** option on the FORM Menu.

2. If there is no current database, the CHOOSE DATABASE Screen appears. After you select a database, the GENERATE FORM Screen is displayed. Enter the name you want to assign to the form (for example, **newform**). **INFORMIX-SQL** asks you for the names of the tables whose columns you want in your form. The GENERATE FORM Menu allows you to enter up to eight tables. (If you want to include more than eight tables in your form specification, use the **New** option on the FORM Menu to create it from scratch.) When you have selected all the tables you want to include, **FORMBUILD** creates a default form specification file. The FORM Menu is then displayed. The default screen form can now be used with **PERFORM**.

   The default form specification file formats the screen as a list of all the columns in the tables included in the form. It does not provide any special instructions to **PERFORM** about how to display the data, nor does it include instructions to perform data manipulations.

3. Select the **Modify** option on the FORM Menu, and INFORMIX-SQL displays the MODIFY FORM Screen. Indicate the name of the default form specification (**newform**). If you have not specified an editor previously in this session or set the DBEDIT environment variable (as explained in Appendix B), INFORMIX-SQL asks for the name of your editor. Then INFORMIX-SQL calls your system editor with the file.

   Edit the default form specification file to produce your customized screen form and associated instructions. Exit from the editor.

4. The MODIFY FORM Menu is displayed. Select the **Compile** option.

5. If your form specification file compiles correctly, a message to that effect is displayed, and FORMBUILD creates a form file with the filename extension **.frm** (for example, **newform.frm**). Go to Step 7. If your form specification file is found to contain errors, a message to that effect is displayed, and FORMBUILD creates a form file with the filename extension **.err** (for example, **newform.err**). Go to Step 6.

6. Select the **Correct** option from the COMPILE FORM Menu. INFORMIX-SQL calls your system editor with the form specification file marked with the compilation errors. When correcting your errors, you need not delete the error messages. INFORMIX-SQL does that for you. Repeat Step 4.

7. When the compilation is successful, select the **Save-and-exit** option on the MODIFY FORM Menu.

As an alternative to using the **Generate** option and creating a default form specification, you can select the **New** option. INFORMIX-SQL calls your system editor.

Because, with the **New** option, you must enter all form specification instructions into the file, the **Generate** option is usually a more efficient way to create a custom form.

Using the Operating System to Create a Form

To create a customized screen form directly from the operating system command line, follow these steps:

1. Create a default form specification file by entering the command

   ```
   sformbld -d
   ```

   at the operating system prompt. **FORMBUILD** asks for the name of your form specification file, the name of your database, and the names of the tables whose columns you want in your form. **FORMBUILD** allows you to enter up to 14 tables. When you enter a blank line for the name of a table, **FORMBUILD** assumes you have selected all the tables and creates a default form specification file. **FORMBUILD** appends the extension **.per** to the name of the file.

   Alternately, you create a form specification file using a system editor. Be sure to name the file with the **.per** extension. If you use this method, proceed to Step 3.

2. Use a system editor to modify the default form specification file to meet your specifications.

3. Enter the command

   ```
   sformbld newform
   ```

   where **newform** is the name of your form specification file (without the **.per** extension). If the compilation is successful, **FORMBUILD** creates a compiled form file called **newform.frm** and you are finished creating your customized screen form. If not, **FORMBUILD** creates a file named **newform.err**, and you need to proceed to Step 4.

4. Edit the file **newform.err** and correct the compilation errors. You must erase the error messages. Overwrite the file **newform.per** with this corrected version and repeat Step 3.

You can run the compiled form specification directly from the command line by entering the command

```
sperform newform
```

You can also create a customized screen form from the operating system command line using a shortened version of the INFORMIX-SQL Main Menu options. Appendix M discusses this method in detail.

Structure of a Form Specification File

Form specification files consist of four required sections (DATABASE, SCREEN, TABLES, and ATTRIBUTES) and one optional section (INSTRUCTIONS):

- **DATABASE Section**: Each form specification file must begin with a DATABASE section identifying the database you want to use with the form.

- **SCREEN Section**: The SCREEN section appears next and shows the exact layout of the form as you want it to appear on the screen. If the form has several screens, this section includes the layout for each screen, one after another.

- **TABLES Section**: Each form specification file must contain a TABLES section following the SCREEN section. The TABLES section identifies the tables whose columns appear in the form.

- **ATTRIBUTES Section**: The ATTRIBUTES section describes each field on the form—including, for example, appearance, acceptable input values, displayed comments, and default values.

- **INSTRUCTIONS Section**: The INSTRUCTIONS section is optional and specifies master-detail relationships, composite joins, alternate field delimiters, and control blocks.

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.

Figure 3-1 illustrates the overall structure of a form specification:

```
database stores

screen
{
--------------------------------------------------------------------------------
CUSTOMER INFORMATION:
Customer Number: [c1          ]          Telephone: [c10               ]
     .
     .
     .

SHIPPING INFORMATION:
     Customer P.O.: [o20        ]

          Ship Date: [o21        ]          Date Paid: [o22        ]
}

end

tables
customer orders

attributes
c1 = *customer.customer_num
   = orders.customer_num;
c10 = phone, picture = "###-###-####x#####";
     .
     .
     .
o20 = po_num;
o21 = ship_date;
o22 = paid_date;

instructions
customer master of orders;
orders master of items;

end
```

Figure 3-1. A Partial Form Specification File

DATABASE Section

The DATABASE section of a form specification file identifies the database with which the form is designed to work.

The DATABASE section consists of the DATABASE keyword followed by the name of the database. The structure of the DATABASE section is

---

DATABASE
    database-name [WITHOUT NULL INPUT]

Notes

1. The WITHOUT NULL INPUT option should be used only if you are working with a database that does not allow NULL values. (See Appendix I for the steps required to create a database without NULL input.) The effect of this option is to cause **INFORMIX-SQL** to display as defaults zeros for numeric and DATE fields and blanks for CHAR fields that do not have alternate defaults.

Examples

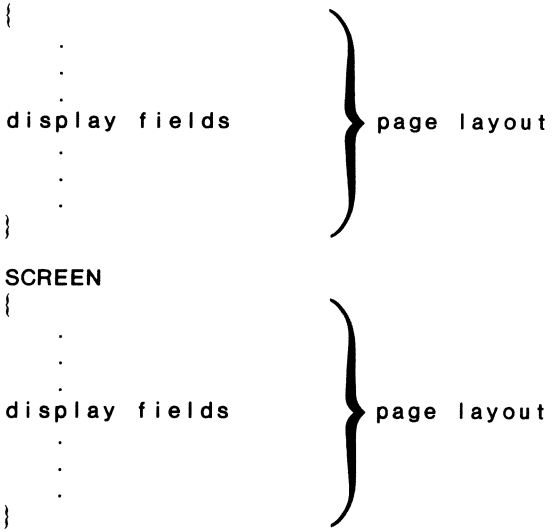
The following DATABASE section is from the **sample** form specification file at the end of this chapter:

```
database
    stores
```

SCREEN Section

The SCREEN section of the form specification file describes how the form will appear on the screen when you use it with **PERFORM.**

The SCREEN section can consist of multiple page layouts. Each page layout is preceded by the SCREEN keyword and is enclosed between { } braces. A page layout consists of an array of *display fields* and textual information, such as titles and field labels. Display fields are indicated by [ ] brackets that define the field length and by field tags that identify the field.



Notes

1. If you include more than 20 screen lines between a pair of braces, FORMBUILD splits the page into two, with line 21 at the top of the second page.

Examples

The **sample** form included at the end of this chapter demonstrates the SCREEN sections of a multiple-page form.

Display Fields

Overview

You indicate where data is to be displayed on the screen by using [ ] brackets to delimit a field. Each field has an associated field tag that identifies the field in the ATTRIBUTES and INSTRUCTIONS sections.

Syntax

---

text [*field-tag*]

Explanation

text	is the material you want displayed on the screen.
[ ]	are delimiters for a field. The width of the field is the number of characters that can be placed between the brackets. In this context, the brackets do not signify an optional input.
field-tag	is the field tag used to identify the display field.

Notes

1. Each field must have a field tag.

2. The field tag is from 1 to 50 characters long. The first character must be a letter; the rest of the tag can include letters, numbers, and underscores (_). The field tag must be short enough to fit within the brackets.

   The same field tag can be used at more than one position in the screen section of the form specification if you want

the same column information to appear in more than one place.

3. Field tags are not the same as database column names; they are the associations used in the SCREEN section, the ATTRIBUTES section, and the INSTRUCTIONS section to tell **PERFORM** where to display and to store information. The ATTRIBUTES section associates each field tag with a column in your database or identifies it as a display-only field.

4. **FORMBUILD** ignores the case of a field tag; **a1** and **A1** are the same.

5. One-character columns are given the display tags **a** through **z**. This means that a screen form can use no more than 26 one-character columns.

6. When you create a default form specification file, the widths of all fields are determined by the data type of the corresponding columns in the database tables.

7. The width of a field in the SCREEN section of the form specification is normally set equal to the width of the database column to which it corresponds. You can reduce or expand the screen width, but should be careful as this can truncate the screen presentation or database storage of the data.

8. Fields corresponding to numeric columns should be large enough to contain the largest number that you might display. If the field is too small to display the number you assign to it, **PERFORM** fills the field with asterisks.

9. Fields for CHAR type data can be shorter than the column's defined length. **PERFORM** will fill the field from the left and truncate the right end of any longer CHAR string assigned to the field. Through subscripting, you may assign portions of a CHAR column to one or more fields (see the "ATTRIBUTES Section," later in this chapter).

10. If you edit and modify the default form specification file or create one from scratch, you can verify that the character column field widths match the data type of the corresponding columns by using the verify (**−v**) option of **FORMBUILD**. Enter the following command in response to the system prompt:

```
sformbld −v newform
```

FORMBUILD reports any discrepancies located in the file **newform.err**.

11. The | bar symbol can be used to denote the close of one field and the beginning of the next field. For example,

```
Name [tag1      |tag2      ]
```

tag1 identifies the first display field; tag2 identifies the second display field.

When using the bar symbol to denote the close of one field and the beginning of the next field, you must include a DELIMITERS statement in the INSTRUCTIONS section of the form specification. Use the same symbol as both the left and right delimiter in the statement.

Examples

The screen layout from the **sample** form specification file follows.

```
screen
{
=============================================================================
=============================================================================

Customer Number: [c1          ]

        Company: [c4                    ]
     First Name: [c2          ]              Last Name: [c3              ]

        Address: [c5                    ]
                 [c6                    ]

           City: [c7              ] State: [c8]   Zip: [c9    ]
      Telephone: [c10               ]


=============================================================================
=============================================================================
}

screen

{
=============================================================================
CUSTOMER NUMBER: [c1          ]          COMPANY: [c4                    ]

ORDER INFORMATION:
     Order Number: [o11          ]      Order Date: [o12           ]

     Stock Number: [i13          ]     Manufacturer: [i16]
      Description: [s14             ]                [m17             ]
             Unit: [s16             ]
                                          Quantity: [i18          ]
                                         Unitprice: [s15                ]
                                       Total Price: [i19                ]

SHIPPING INFORMATION:
 Customer P.O.: [o20          ]        Ship Charge: [d1          ]

       Backlog: [a]             Total Order Amount: [d2          ]
     Ship Date: [o21          ]
     Date Paid: [o22          ]
  Instructions: [o23                                 ]

}
end
```

TABLES Section

Overview

The third section of the form specification file lists all the tables from which columns will be drawn to appear in the screen form. You need not display in the screen form every column of every table listed, but any table contributing data to the form must be included.

The TABLES section consists of the TABLES keyword, followed by a table name or a list of table names separated by spaces. The number of tables that you can use in a form is machine dependent. On most UNIX systems, the maximum number of tables open at one time is 12; on most DOS systems, the maximum number of tables open at one time is 10.

The structure of the TABLES section is shown below.

TABLES *table* . . .

Examples

The TABLES section from the **sample** form specification file at the end of this chapter is

---

```
tables
    customer items stock
    orders manufact
```

Notes

1. You may not include a temporary table in your table list.

2. You can build a form based on a view as long as the columns contributing data to the view belong to only one table. Aggregate data is not allowed.

ATTRIBUTES Section

Overview

The ATTRIBUTES section describes the behavior and appearance of each field defined in the SCREEN section. Every field in the SCREEN section must be described in the ATTRIBUTES section. You use *attributes* to describe how PERFORM should display the field, to specify a default value, to limit the values that may be entered, and to set other parameters as described in the "ATTRIBUTES Syntax" section.

The order in which the fields are described in the ATTRIBUTES section determines the default order for the cursor movement on the screen. The order in which columns are referenced determines the order in which PERFORM makes tables active (that is, available for data entry).

Fields in the SCREEN section do not have to be associated with a column. A field not associated with a column is called a *display-only field.* The ATTRIBUTES section contains two kinds of *link statements*: those linking field tags to database columns and those linking field tags to display-only fields.

Display Field Order

When you use the **Query**, **Add**, and **Update** options in PERFORM, the cursor advances by default from field to field in the active table according to the order in which the field tags appear in the ATTRIBUTES section of the form specification. When the cursor has advanced through all the fields in the active table on the screen, it returns cyclically to the first field. The default order may be changed by the use of control blocks, described later in this chapter in the "INSTRUCTIONS Section."

Table Order

When a form contains fields corresponding to several database tables, **PERFORM** puts the tables in an ordered list. When you use the **Table** option, **PERFORM** changes the active table by selecting the next table in the list. **PERFORM** assigns tables in the order in which columns in the tables are referenced in the ATTRIBUTES section. You reference a table whenever you associate a column from the table with a tag name or to another column in a join.

Fields Linked to Database Columns

Overview

There are two kinds of fields that are linked to database columns in the ATTRIBUTES section: those that accept input and those that do not. Fields that are linked to database columns and that do not allow input from the keyboard are called *lookup fields*. You specify lookup fields with the LOOKUP attribute, defined later in this chapter in the "ATTRIBUTES Syntax" section. This section describes column-linked fields that allow input.

Syntax

---

field-tag = [*table.*]*col*[, *attr*];

Explanation

field-tag is the field tag used in the SCREEN section.

table is the name of a database table. You need to specify *table* only if the same *col* name occurs in more than one table in the form. (This allows **PERFORM** to uniquely identify a column.)

col is the name of a column.

attr is a **FORMBUILD** attribute or a list of attributes separated by commas.

Notes

1. Only the field tag, the equal sign, the column name, and the semicolon are required.

2. Portions of CHAR type columns can be displayed in a field by using subscripting. For example, the **orders** table has a **ship_instruct** column that is a CHAR type column of length 40. You can display it on the screen as two display fields of length 20. If the field tags for the two fields are **inst1** and **inst2**, respectively, the ATTRIBUTES section entry is

```
inst1 = ship_instruct[1,20];
inst2 = ship_instruct[21,40];
```

Overview

Display-only fields are not associated with columns of the database and appear only on the screen. They receive their values as a result of calculations and/or logical decisions based on the values in other fields.

Syntax

---

field-tag = DISPLAYONLY [ALLOWING INPUT]
TYPE *datatype* [NOT NULL] [, *attr*];

Explanation

field-tag	is the field tag used in the SCREEN section.
DISPLAYONLY	is the keyword indicating that the field does not correspond to a column of a table in the database. You describe how such a field will receive its value in the INSTRUCTIONS section.
ALLOWING INPUT	are the keywords that you use to allow input.
TYPE	is a required keyword.
datatype	is any one of the data types permitted by **INFORMIX-SQL** except SERIAL (see Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide* for information about data types).

NOT NULL	are optional keywords informing **INFORMIX-SQL** that, if the field allows input, the user must give it a value.
attr	is an attribute or a list of attributes separated by commas.

Notes

1. Do not give a length to type CHAR; the length is determined by the display width.

2. If you specify the precision for a DECIMAL or MONEY type, be certain that the display width can hold the value.

3. When the field does not allow input, you may use only the following attributes with display-only fields:

DEFAULT	DOWNSHIFT
FORMAT	QUERYCLEAR
REVERSE	RIGHT
UPSHIFT	ZEROFILL

4. When you specify that one or more display-only fields allow input, **PERFORM** collects these fields into a database table named **displaytable**. No database table is actually created, but **PERFORM** behaves as though **displaytable** existed and as though its field tags were column names. You use **displaytable** in the INSTRUCTIONS section to control data entry and cursor movement in display-only fields allowing input.

5. If a **displaytable** exists, it is always the last table in the sequence of active tables, regardless of how you order its fields among the other field tags.

6. When the DATABASE section has the WITHOUT NULL INPUT clause, the NOT NULL keywords instruct **INFORMIX-SQL** to use zero (numeric data type) or blanks (CHAR data type) as a default for this field.

Examples

The ATTRIBUTES section of the **sample** form specification file contains two DISPLAYONLY fields.

```
d1 = displayonly type money;
d2 = displayonly type money;
```

These fields are used to calculate the shipping charge and total order amount for each order. This information is not stored in any columns in the database.

Joining Columns

A screen form that contains information from several database tables normally includes a display field that *joins* two (or more) database columns containing the same information. While it is not required that the join columns be indexed, it is advisable since cross-table queries will not run as quickly if the underlying join columns are not indexed.

The database columns you join must be of the same data type. If they are CHAR columns, they must be the same length. Do not join two SERIAL columns to each other; join a SERIAL column only to an INTEGER column.

You join columns by equating them to the same field tag in the ATTRIBUTES section:

---

field-tag = col1 = col2;

An example from the **sample** form is

```
o11 = *orders.order_num = items.order_num;
```

Field-tag **o11** joins the **order_num** column of the **orders** table with the **order_num** column of the **items** table. (The asterisk placed before the **orders.order_num** column name indicates that this is a special kind of join—a verify join. Verify joins are explained in the next section.)

The placement of attributes determines when they take effect. If you want an attribute to apply regardless of which table in the join is active, place the column names on the same line and the attribute after the last column name:

---

field-tag = col1 = col2, attr;

If you want different attributes to apply for each of the columns in the join, place the column names on separate lines:

```
field-tag = col1, attr1;
          = col2, attr2;
```

attr1 is effective when the table containing *col1* is active, and *attr2* is effective when the table containing *column2* is active.

Here is an example from the **sample** form:

```
i13 = items.stock_num;
    = *stock.stock_num, noentry,
       noupdate, queryclear;
```

The attributes NOENTRY, NOUPDATE, and QUERYCLEAR (explained in the "ATTRIBUTES Syntax" section) are effective only when the **stock** table is active.

The FORMAT and REVERSE attributes, also described later, always take effect, regardless of their placement.

Verify Joins

You can verify that the value you enter into a field corresponding to a column in one table already exists in another column (the *dominant* column) in another table. You do this through a *verify join*. The verify join is indicated by placing an asterisk in front of the dominant column name:

```
field-tag = col1 = *col2;
```

PERFORM will prevent entry of any value into *field-tag* that does not already occur in *col2*. (This applies for non-composite conditions.)

For example, when assigning orders to customers, you want to ensure that the customer number entered for a store is a valid customer number in the **customer** table. The following statement in the ATTRIBUTES section of the **sample** form does just this.

```
c1   = *customer.customer_num
     = orders.customer_num;
```

In the above statement, the **customer.customer_num** column is the dominant column in a verify join; you may not enter a value into field **c1** when the **orders** table is active that does not already exist in the **customer_num** column of the **customer** table.

A third kind of join, described later in the description of the LOOKUP attribute, allows you to display or to verify data from a table that is not active.

ATTRIBUTES Syntax

PERFORM recognizes the attributes listed below. The syntax for each attribute is detailed in the following sections.

AUTONEXT	PICTURE
COMMENTS	QUERYCLEAR
DEFAULT	REQUIRED
DOWNSHIFT	REVERSE
FORMAT	RIGHT
INCLUDE	UPSHIFT
LOOKUP	VERIFY
NOENTRY	ZEROFILL
NOUPDATE	

AUTONEXT

Overview

Use the AUTONEXT attribute to cause the cursor to advance automatically to the next field when the current field is full.

Syntax

---

field-tag = *col*, AUTONEXT;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
AUTONEXT	is a keyword that tells **PERFORM** to advance the cursor to the next field when the current field is full.

Notes

1. AUTONEXT is particularly useful for entering text into a CHAR type database column that is split among two or more display fields with the use of subscripts.

2. Another use of AUTONEXT is with CHAR fields in which the input data is of a standard length (for example, the abbreviation for a state name is always two digits) or when the CHAR field has a length of one (only one keystroke is required to enter the data and to move to the next field).

Examples

The **sample** form specification file uses the AUTONEXT attribute in displaying the **state** and **zipcode** columns from the **customer** table:

```
c8 = state, upshift, autonext;
c9 = zipcode, autonext;
```

When two characters are entered into the **c8** field (and the field is full), the cursor moves automatically to the beginning of the next field (the **c9** field). When five characters are entered into the **c9** field (and the field is full), the cursor moves automatically to the beginning of the next field.

COMMENTS

Overview

Use COMMENTS to cause **PERFORM** to display a message on the Comment Line at the bottom of the screen. The message is displayed when the cursor moves to the associated field.

Syntax

---

field-tag = *col*, COMMENTS = "*message*";

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
COMMENTS	is a required keyword.
message	is a character string enclosed in quotes.

Notes

1. The *message* must appear in quotation marks on a single line of the form specification file.

2. The Status Line is the bottom line of the screen. The Comment Line is just above the Status Line.

3. The most common use of the COMMENTS attribute is to give information or instructions to the user. This is particularly appropriate when the field will accept only a limited set of user-specified values.

Examples

An example from the **sample** form specification file follows:

```
c2 = fname, comments =
   "Please enter initial if available." ;
```

Related Attributes

INCLUDE

DEFAULT

Overview

Use the DEFAULT attribute to assign a default value to a display field.

Syntax

---

field-tag = *col*, DEFAULT = *value*;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
DEFAULT	is a required keyword.
value	is the default value.

Notes

1. If you do not use the DEFAULT attribute, display fields default to blanks.

2. Enclose DATE values and CHAR values that contain spaces or special characters in quotation marks. Using quotation marks around CHAR values that contain no spaces or special characters is optional.

3. PERFORM displays the default value whenever the field is displayed for data entry in an **Add** operation.

4. If both the DEFAULT attribute and the REQUIRED attribute are assigned to the same field, the REQUIRED attribute is ignored.

5. Use the TODAY keyword as the *value* to assign the current date as the default value of a field.

Examples

Two examples from the **sample** form specification file follow:

```
c8   = state, upshift, autonext,
          default = "CA" ;

o12 = order_date, default = today,
          format = "mm/dd/yyyy" ;
```

DOWNSHIFT

Overview

Assign the DOWNSHIFT attribute to a CHAR field when you want **PERFORM** to convert uppercase letters to lowercase letters.

Syntax

---

field-tag = *col*, DOWNSHIFT;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
DOWNSHIFT	is the keyword that instructs **PERFORM** to convert CHAR input data to lowercase.

Notes

1. Because upper- and lowercase letters have different ASCII values, storing character strings in one format or the other can simplify sorting and querying a database.

Related Attributes

UPSHIFT

FORMAT

Overview

Use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT, or DATE column to control the format of the display.

Syntax

---

field-tag = *col*, FORMAT = "*fstring*";

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
FORMAT	is a required keyword.
fstring	is a (format) string of characters that specifies the desired data format. You must enclose *fstring* in quotation marks.

Notes

1. For DECIMAL, SMALLFLOAT, or FLOAT data types, *fstring* consists of pound signs (#) that represent digits and a decimal point. For example, ###.## produces up to three places to the left of the decimal point and exactly two places to the right.

2. If the actual displayed number is shorter than the *fstring*, **PERFORM** right-justifies it and pads the left with blanks.

3. If the *fstring* is smaller than the display width, **FORMBUILD** gives a warning, but the form is usable.

4. If necessary, **PERFORM** rounds numbers before displaying them.

5. For DATE data types, **PERFORM** recognizes the following symbols as special in the string *fstring*:

   mm produces the two-digit representation of the month.

   mmm produces a three-letter abbreviation of the month, for example, **Jan**, **Feb**, and so on.

   dd produces the two-digit representation of the day.

   ddd produces a three-letter abbreviation of the day of the week, for example, **Mon**, **Tue**, and so on.

   yy produces the two-digit representation of the year.

   yyyy produces a four-digit year.

   For dates, **FORMBUILD** interprets any other characters as literals and displays them wherever you place them within *fstring*.

Examples

Example FORMAT attributes for DATE fields include:

Input	Result
no FORMAT attribute	09/15/1984
FORMAT = "mm/dd/yy"	09/15/84
FORMAT = "mmm dd, yyyy"	Sep 15, 1984
FORMAT = "yymmdd"	840915
FORMAT = "dd-mm-yy"	15-09-84
FORMAT = "(ddd.) mmm. dd, yyyy"	(Sat.) Sep. 15, 1984

Two examples from the **sample** form specification file follow:

```
o12 = order_date,
          default = today,
          format = "mm/dd/yyyy";

o22 = paid_date,
          format = "mm/dd/yyyy";
```

Related Attributes

PICTURE

INCLUDE

Overview

Use the INCLUDE attribute to specify acceptable values for a field and to cause **PERFORM** to check input before accepting it.

Syntax

---

field-tag = *col*, INCLUDE = (*val-list*);

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
INCLUDE	is a required keyword.
val-list	is either a list of individual values (*value1*, *value2*, ...) or a range of values (*value1 TO value2*) or a combination of individual values and ranges separated by commas.

Notes

1. When you specify a range of values, the lower value must appear first.

2. For ranges of character values, **PERFORM** uses dictionary ordering with the printable ASCII character set (see Appendix F for an ordered list of the ASCII character set). In a numeric field, the range (5 to 10) is acceptable. In a CHAR field, it is incorrect. The character string *10* is less than the string *5*, since *1* comes before *5* in the ASCII character set.

3. If you include a character string that contains a blank space, a comma, or any special characters, you must enclose the entire string in quotation marks. It is advisable to enclose character strings in quotation marks at all times.

4. Before **PERFORM** accepts a new row, you must enter an acceptable value in each display field with the INCLUDE attribute. Use the keyword NULL to indicate that a Null value is acceptable. If a field has both the DEFAULT and INCLUDE attributes, then the DEFAULT value must appear in the INCLUDE list. Otherwise the form will not compile.

5. Including a COMMENT indicating acceptable values makes data entry easier.

Examples

An example from the **sample** form specification file follows:

```
i18 = items.quantity,
      include = (1 to 50),
      comments = "Acceptable values are 1 through 50";
```

Related Attributes

COMMENT, REQUIRE

LOOKUP

Overview

Use the LOOKUP attribute to display data from another table while entering data into or querying the active table. You may also use it to prevent data from being entered into the active table if the value does not exist in another table.

Syntax

---

field-tag = *table1.col*, LOOKUP[*field-tag1* = *table2.col1*
[,*field-tag2* = *table2.col2* ,...]]
JOINING [*]*table2.col*;

Explanation

field-tag is the field tag used in the SCREEN section.

table1.col is the name of a column belonging to table *table1*.

LOOKUP is the keyword specifying a join.

field-tag1 is the field tag of a field that will display a value from the LOOKUP table.

table2.col1 is a column in table *table2* whose value is displayed in *field-tag1*.

JOINING is the keyword that identifies the joined column.

* is an optional keyword, identifying the dominant column in a verify join (see the "Verify Joins" section earlier in this chapter).

table2.col is the name of a column belonging to table *table2* and being joined to *table1.col*.

Notes

1. The optional asterisk placed in front of *table2.col* column name tells **PERFORM** to accept a value for *table1.col* only if the same value already exists in *table2.col.*

2. The optional list of identifications of field tags with column names following the LOOKUP attribute directs **PERFORM** to display these values whenever there is a successful join between *table1.col* and *table2.col.* Values cannot be entered into these fields from the keyboard.

3. If the join columns in a LOOKUP are not indexed, the LOOKUP does run as quickly.

Examples

An example LOOKUP JOIN from the **sample** form specification file follows:

```
i16 = items.manu_code,
      lookup m17 = manufact.manu_name
      joining  *manufact.manu_code;
```

In this example the entry of the item manufacturer code number is checked against the list of manufacturer code numbers in the **manufact** table. If the same value is found there, the manufacturer's name is extracted from the **manufact** table and is displayed in field **m17.**

NOENTRY

Overview

Use the NOENTRY attribute to prevent data entry during the creation of a new row during an **Add** operation.

Syntax

---

field-tag = *col*, NOENTRY;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
NOENTRY	is the keyword indicating that no data entry may be made to this field during an **Add** operation.

Notes

1. The NOENTRY attribute does not prevent you from modifying the field during an **Update** operation.

2. The NOENTRY attribute is unnecessary with a SERIAL column.

Examples

Two examples from the **sample** form follow:

```
i13 = items.stock_num;
    = *stock.stock_num, noentry,
       noupdate, queryclear;

s14 = stock.description, noentry,
       noupdate;
```

When the **stock** table is active, the columns **i13** and **s14** (corresponding to the columns **stock.stock_num** and **stock.description**, respectively) cannot have values added. (The inclusion of the **noupdate** attribute prevents data entry during an **Update** operation.)

Related Attributes

NOUPDATE

NOUPDATE

Overview

Use the NOUPDATE attribute to prevent data entry during the modification of a row during an **Update** operation.

Syntax

---

field-tag = *col*, NOUPDATE;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
NOUPDATE	is the keyword indicating that no data entry may be made to this field during an **Update** operation.

Notes

1. The NOUPDATE attribute does not prevent you from entering data into the field during an **Add** operation.

Examples

Two examples from the **sample** form follow:

```
s15 = stock.unit_price, noentry,
          noupdate;

s16 = stock.unit_descr, noentry,
          noupdate;
```

When the **stock** table is active, the fields **s15** and **s16** (corresponding to the columns **stock.unit_price** and **stock.unit_descr**, respectively) cannot receive values during an **Update** operation. (The inclusion of the **noentry** attribute prevents data entry during an **Add** operation.)

Related Attributes

NOENTRY

PICTURE

Overview

Use the PICTURE attribute to specify the character pattern for data entry to a CHAR type field.

Syntax

---

field-tag = *col*, PICTURE = "*pstring*";

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
PICTURE	is a required keyword.
pstring	is a (picture) string of characters that specify the desired character pattern.

Notes

1. *pstring* is a combination of three special symbols:

Symbol	Meaning
A	Any letter
#	Any digit
X	Any character

   Any other character in the *pstring* is treated as a literal and will occur, during data entry, in the exact location indicated.

2. If you attempt to enter a character that does not conform with the *pstring*, the terminal beeps and **PERFORM** does not echo the character on the screen.

3. The PICTURE attribute does not require the entry of the entire field; it only requires that what you enter conforms to *pstring*. Note that the length of *pstring* must equal the length of the corresponding display field.

4. **PERFORM** will remind data entry operators of the required pattern by displaying the literal characters in the display field and leaving blanks elsewhere.

Examples

An example from the **sample** form follows:

```
c10 = phone,
          picture = "###-###-####x####";
```

produces the following display field before data entry:

```
[    -  -           ]
```

As another example, if you specify a field for part numbers like this,

```
f1 = part_no, picture = "AA#####-AA(X)";
```

PERFORM would accept any of the following inputs:

```
LF49367-BB(*)
TG38524-AS(3)
YG67489-ZZ(D)
```

Related Attributes

FORMAT

QUERYCLEAR

Overview

Use the QUERYCLEAR attribute to clear a joining field on the screen when you enter a **Query** operation.

Syntax

---

field-tag = *col*, QUERYCLEAR;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
QUERYCLEAR	is the keyword that instructs **PERFORM** to clear a joining field when you enter a **Query** operation.

Notes

1. When you enter the **Query** option, **PERFORM** normally clears all fields except joining and display-only fields.

2. QUERYCLEAR does not apply to display-only fields. You must give explicit instructions in the INSTRUCTIONS section to clear display-only fields.

Examples

An example from the **sample** form specification file follows:

```
i13 = items.stock_num;
    = *stock.stock_num, noentry,
          noupdate, queryclear;
```

Here the **items** table and the **stock** table are joined through the stock number. When the **stock** table is the active table and a query is made, the **stock_num** field will be cleared. When **items** is the active table, however, the **stock_num** field will not be cleared when a query is made.

REQUIRED

Overview

Use the REQUIRED attribute to force data entry into a particular field during an **Add** operation.

Syntax

---

field-tag = *col*, REQUIRED;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
REQUIRED	is the keyword that instructs **PERFORM** to insist upon data entry to the *field-tag* field.

Notes

1. The REQUIRED attribute has no effect during **PERFORM**'s **Update** operation. You are free to erase values from REQUIRED fields when you use an **Update** operation.

2. There is no default value for a REQUIRED field. If you assign both the REQUIRED attribute and the DEFAULT attribute to the same field, the REQUIRED attribute is ignored.

Examples

An example from the **sample** form follows:

```
o20 = po_num, required;
```

FORMBUILD requires the entry of a purchase order value when adding a new order to the database.

REVERSE

Overview

Assign the REVERSE attribute to fields you want **PERFORM** to display in reverse video.

Syntax

---

field-tag = *col*, REVERSE;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
REVERSE	is the keyword that instructs **PERFORM** to display the *field-tag* field in reverse video.

Notes

1. On terminals that do not support reverse video, fields having the REVERSE attribute are enclosed in angle < > brackets.

RIGHT

Overview

Assign the RIGHT attribute to fields whose data you want right-justified.

Syntax

---

field-tag = *col*, RIGHT;

Explanation

field-tag is the field tag used in the SCREEN section.

col is the name of a database column.

RIGHT is the keyword that instructs **PERFORM** to right-justify the data entered into the *field-tag* field.

Notes

1. **PERFORM** right-justifies data you enter during an **Add** or **Update** operation.

2. To search for a right-adjusted CHAR field of value "*string*" during a **Query** operation, use the wildcard search pattern "**string*" to account for potential leading blanks.

UPSHIFT

Overview

Assign the UPSHIFT attribute to a CHAR field when you want **PERFORM** to convert lowercase letters to uppercase letters.

Syntax

---

field-tag = *col*, UPSHIFT;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
UPSHIFT	is the keyword that instructs **PERFORM** to convert CHAR input data to uppercase.

Notes

1. Because uppercase letters and lowercase letters have different ASCII values, storing character strings in one format or the other can simplify sorting and querying of a database.

Examples

An example from the **sample** form follows:

```
c8   = state, upshift, autonext,
           include = ("CA", "OR", "NV", "WA"),
           default = "CA" ;
```

Because of the UPSHIFT attribute, **PERFORM** enters uppercase characters in the **state** field regardless of the case used to enter them.

Related Attributes

DOWNSHIFT

VERIFY

Overview

Use the VERIFY attribute when you want **PERFORM** to require users to enter data twice for a particular field to reduce the probability of erroneous data entry.

Syntax

---

field-tag = *col*, VERIFY;

Explanation

field-tag is the field tag used in the SCREEN section.

col is the name of a database column.

VERIFY is the keyword that instructs **PERFORM** to require duplicate data entry to the *field-tag* field.

Notes

1. Since some data is critical, the VERIFY attribute supplies an additional step in data entry to ensure the integrity of the data in your database. After you enter a value into a VERIFY field and press **RETURN**, **PERFORM** erases the field and requests that you re-enter the value. You must enter *exactly* the same data each time, character for character: 15000 is not exactly the same as 15000.00.

Examples

For example, if you specify a field for salary information like this,

```
s10 = salary, verify;
```

PERFORM requires the entry of exactly the same data twice.

ZEROFILL

Overview

Assign the ZEROFILL attribute to fields that you want to be right-justified and padded with leading zeros.

Syntax

---

field-tag = *col*, ZEROFILL;

Explanation

field-tag	is the field tag used in the SCREEN section.
col	is the name of a database column.
ZEROFILL	is the keyword that instructs **PERFORM** to right-justify fields and pad with leading zeros.

Notes

1. This attribute is most useful with numeric fields. If the number entered into the field is shorter than the field itself, **PERFORM** right-justifies it and fills the leading blanks with zeros.

INSTRUCTIONS Section

The final section of the form specification file is the optional INSTRUCTIONS section. This section is used to

- Establish composite joins
- Specify alternate field delimiters
- Create master/detail relationships
- Define control blocks

You can also call C functions from within the INSTRUCTIONS Section if you have **INFORMIX-ESQL/C**. See the *INFORMIX-ESQL/C Programmers Manual* for details.

The INSTRUCTIONS section begins with the INSTRUCTIONS keyword.

COMPOSITES

Overview

Establish a *composite join* between two tables when you must specify the values of more than one column in a table to specify a row uniquely.

Syntax

COMPOSITES <*table1.col1*, *table1.col2*[, *table1.col3*[, ...]]>
[*]<*table2.col1*, *table2.col2*[, *table2.col3*[, ...]]>;

Explanation

COMPOSITES	is the keyword indicating that the following sets of column names enclosed in angle < > brackets are to be treated as composite columns that are joined to each other.
table1.colJ	(where J = 1, 2, 3, ...) is a column in *table1*.
table2.colJ	(where J = 1, 2, 3, ...) is a column in *table2*.
*	indicates that the join is a verify join—that is, unless the marked composite exists in *table2*, **PERFORM** will not allow the corresponding row to be written to *table1*.

Notes

1. Each column included in a composite join must also be individually joined in the ATTRIBUTES section of the form specification. This means that *table1.col1* must be joined individually to *table2.col1* in the ATTRIBUTES section, as must *table1.col2* to *table2.col2*, and so on.

2. There can be no additional joins between columns of the two tables that are not included in the composite join.

3. If the columns in a composite join are not individually and jointly indexed, cross-table queries do not run as quickly.

Examples

An example from the **sample** form follows:

```
composites <items.stock_num, items.manu_code>
         * <stock.stock_num, stock.manu_code>
```

The **stock_num** and **manu_code** fields in the **items** and **stock** table are included in a composite join. This is a composite verify join. When the **items** table is active, values entered in the **stock_num** and **manu_code** fields are compared with values existing in those two columns in the **stock** table. **PERFORM** will notify the user if there is not a match and reject the entry. This precludes the entry of stock numbers and manufacturer codes that individually exist in the database but, as a composite, do not correspond to a unique row in the **stock** table.

To specify a unique row in the **stock** table requires both the **stock_num** and **manu_code**. For example, the **stock** table contains three rows with the stock number 1, and four rows with the manufacturer code HRO. (See Appendix A for a list of data included in the sample database.) Knowing the stock number or manufacturer code alone does not allow you to locate a unique row. You need both the stock number (1) and the manufacturer code (HRO) to specify a unique row (baseball gloves produced by Hero) in the table.

DELIMITERS

Overview

You may change the delimiters that **PERFORM** uses to enclose the fields when the form appears on the screen. The default delimiters are [ ] brackets, but you may substitute any other printable character, including blank spaces.

Syntax

DELIMITERS "*ab*";

Explanation

DELIMITERS is a required keyword.

a is the opening delimiter.

b is the closing delimiter.

Notes

1. The DELIMITERS instruction tells **PERFORM** the symbol to use as a delimiter when it displays the fields on the screen.

2. Each delimiter is a single character only.

3. **FORMBUILD** still requires that you use brackets in the form specification file.

4. If your form has columns from more than one database table, you may not want to use blank spaces as delimiters. If you do use blank spaces, you will have no visual indication on the screen of which fields correspond to columns in the active table.

5. You may use the | bar symbol to denote both a closing delimiter and an opening delimiter. For example,

```
Name [tag1      |tag2      ]
```

tag1 identifies the first display field; tag2 identifies the second display field. If you use the bar symbol in the SCREEN section, you must include a DELIMITERS statement in the INSTRUCTIONS section of the form specification. Use two identical symbols for the left and right delimiter in the DELIMITERS statement.

MASTER OF

Overview

Create a master/detail relationship between two tables when a row of one table (master) is associated with several rows of another table (detail).

Syntax

---

table1 MASTER OF *table2*;

Explanation

table1	is a table in the database that is designated as the master table.
MASTER OF	are required keywords.
table2	is a table in the database that is designated as the detail table.

Notes

1. The master/detail relationship simplifies cross-table queries, especially when one row of *table1* is associated with several rows of *table2*.

2. Master/detail relationships can be defined in both directions.

3. If no explicit master/detail relationship exists, **PERFORM** displays an error message when you use the **Master** or **Detail** option.

Examples

Two examples from the **sample** form specification file follow:

```
customer master of orders;
orders master of items;
```

These master/detail relationships are useful because each customer can have many orders, and each order can have many items.

Additional examples are displayed in the following table.

Master	Detail
projects	personnel
orders	items
agents	clients
parents	children

These master/detail relationships are useful where several staff members (**personnel**) work on the same project (**projects**), each purchase order (**orders**) contains more than one item (**items**), or a single agent (**agents**) has many clients (**clients**).

With a master/detail relationship defined in both directions, you can explore the database in the following way. Suppose you have a database consisting of **personnel** and **projects** tables. Each person is assigned to a single project, and each project has several people working on it. The screen form includes an INSTRUCTION section stating

```
personnel master of projects;
projects master of personnel;
```

Assume you want to query the database to find all employees who work with a particular employee, but you do not know on which project they work. When you identify and bring to the screen the particular employee (the **personnel** table is active) and select the **Detail** option, **PERFORM** moves to the PROJECT INFORMATION Screen (the **projects** table is active) and displays the information on the employee's project. If you then choose the **Detail** option, **PERFORM** selects all employees on that project and shifts to the PERSONNEL INFORMATION Screen (the **personnel** table is active).

Control Blocks

Use *control blocks* to perform these functions:

- Control the cursor movement when you **Add** or **Update** a row.
- Check the value of data you enter against criteria that depend upon other data that has already been entered.
- Modify the data in fields after **Add**, **Update**, and **Query** operations.
- Perform calculations on field values and enter the results into another field.
- Display aggregate information like averages and totals on columns in the current list. (The current list is the set of rows that results from a **Query** as modified by subsequent **Add** or **Remove** actions.)
- Call C functions from **PERFORM** if you have **INFORMIX-ESQL/C**. See the *INFORMIX-ESQL/C Programmers Manual* for details.

Each control block is either a BEFORE block or an AFTER block. Screen control actions can be taken either before or after **PERFORM** operations are completed. You may use BEFORE blocks with the **Add, Update,** and **Remove** operations. You may use AFTER blocks with the **Add**, **Update**, **Query**, and **Remove** operations.

BEFORE

Overview

Use a BEFORE control block to cause **PERFORM** to take a series of actions before it executes an operation.

Syntax

BEFORE *option-list* OF *table/column-list*

action
action
.
.
.
action

Explanation

BEFORE	is a required keyword.
option-list	is one or more of the keywords EDITADD, EDITUPDATE, or REMOVE separated by spaces. These keywords are defined below.
OF	is a required keyword.
table/column-list	is a list of up to sixteen names of database tables and/or columns. Depending upon your operating system, the limit on the number of table names may be lower.

action is one of five kinds of instructions to **PERFORM**: assign values to fields, move the cursor to a field, display a message, exit to the **PERFORM** Menu, or take one of these actions depending upon conditions among the values in the fields. These actions are defined in the "Action Syntax" section later in this chapter.

Notes

1. The EDITADD and EDITUPDATE keywords refer to the act of editing during an **Add** and an **Update,** respectively. For EDITADD and EDITUPDATE, the actions are taken before **PERFORM** writes the row to the table.

2. You can use the BEFORE REMOVE operation with the ABORT keyword (described later in this chapter) to prevent a user from removing the last row from a detail table.

AFTER

Overview

Use an AFTER control block to cause **PERFORM** to take a series of actions after it executes an option.

Syntax

AFTER *option-list* OF *table/column-list*

action
action
.
.
.
action

Explanation

AFTER	is a required keyword.
option-list	is one or more of the keywords ADD, UPDATE, QUERY, REMOVE, DISPLAY, EDITADD, and EDITUPDATE, separated by spaces. These keywords are defined below.
OF	is a required keyword.
table/column-list	is a list of up to sixteen names of database tables and/or columns. Depending upon your operating system, the limit on the number of table names may be lower.

action is one of four kinds of instructions to **PERFORM**: assign values to fields, move the cursor to a field, display a message, or take one of these actions depending upon conditions among the values in the fields. These actions are defined in the "Action Syntax" section later in this chapter.

Notes

1. The ADD, UPDATE, QUERY, and REMOVE keywords correspond to the **PERFORM** options **Add**, **Update**, **Query**, and **Remove**, respectively. See Chapter 4, "The PERFORM Screen Transaction Processor," for more information about the **PERFORM** options.

2. The DISPLAY keyword refers to the display of fields after the execution by **PERFORM** of the **Next**, **Previous**, **Query**, or other options.

3. EDITADD and EDITUPDATE differ from ADD and UPDATE. For EDITADD and EDITUPDATE, the actions are taken before **PERFORM** writes the row to the table. For ADD and UPDATE, they are taken after **PERFORM** writes the row to the table.

4. You can list only table names, including **displaytable**, in the *table/column-list* following the ADD, UPDATE, QUERY, REMOVE, and DISPLAY keywords.

EDITADD and EDITUPDATE

Overview

The EDITADD and EDITUPDATE keywords give you the ability to perform one or more actions before or after you enter data into a field during an **Add** and an **Update** operation, respectively. The action occurs before the row is written to the table.

Syntax

{BEFORE | AFTER} {[EDITADD] [EDITUPDATE]}
OF *table/column-list*

action
action
.
.
.
action

Explanation

BEFORE	is a keyword.
AFTER	is a keyword.
EDITADD	is a keyword referring to the act of editing during an **Add**.
EDITUPDATE	is a keyword referring to the act of editing during an **Update**.
table/column-list	is a list of up to sixteen names of database tables and/or columns. This number is machine dependent; the limit on the number of table names may be lower on your system.

Notes

1. If the *table/column-list* contains the names of columns only, the BEFORE keyword instructs **PERFORM** to execute the actions when the cursor moves to the corresponding field, before you enter data.

2. If the *table/column-list* contains the names of columns only, the AFTER keyword instructs **PERFORM** to execute the actions when you enter data into the corresponding field and press **RETURN**. **PERFORM** will make all the attribute-specified checks (such as INCLUDE, VERIFY, and so on) before executing the actions.

3. You must enter the names of database columns in the *table/column-list*. Do not use the field tags.

4. When you specify a database table instead of a column in a BEFORE block, **PERFORM** executes the actions before you enter any data into the form. Using this feature, you can make **PERFORM** enter defaults into fields and display comments depending upon the active table.

5. When you specify a database table instead of a column in an AFTER block, **PERFORM** executes the actions after you enter all the data and press **ESCAPE** to complete the transaction, before the row is written to the database. Using this technique, you can make consistency checks of all the data entered and return to data entry if you find inconsistencies.

6. In a BEFORE block, when you refer to a CHAR column that is split into more than one field, **PERFORM** will execute the actions before each section of the displayed field. If you want these actions executed only before the first section of a split field, replace the BEFORE block of the split field with an AFTER block of the immediately preceding field.

7. If you want the actions executed only after the last section of a split field, replace the AFTER block of the split field with a BEFORE block of the immediately succeeding field.

Examples

The following examples are taken from the **sample** form specification file at the end of this chapter. The syntax of the action statements used in these examples is described later.

```
after editadd editupdate of quantity
   let i19 = i18 * s15
   nextfield = o11
```

After you enter a value in the **quantity** field (using the **Add** or **Update** options), **PERFORM** calculates and places the value in the **i19** (Total Price) column, and places the cursor in the **o11** (Order Number) field.

```
before editadd editupdate of orders
   nextfield = o20
```

In this example, as soon as you indicate that you want either the **Add** or **Update** options when **orders** is the active table, **PERFORM** is instructed to move the cursor to the **o20** (Customer P.O.) field. Without this instruction, the cursor would go first to the **o11** (Order Number) field, since it is the first **orders** field to appear in the ATTRIBUTES section of the form.

ADD

Overview

Use the ADD keyword to cause **PERFORM** to execute actions after the **Add** operation. The action occurs after the row is written to the table.

Syntax

AFTER ADD OF *table-list*

action
action
.
.
.
action

Explanation

AFTER ADD OF	are required keywords.
table-list	is a space-separated list of database tables.

Notes

1. The main use of the ADD keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

Examples

The following example is from the **sample** form. The action statements used in this example are described in the "Action Syntax" section of this chapter.

```
after add update query of items
   if (total of i19) <= 100 then
      let d1 = 7.50
      else
      let d1 = (total of i19) * .04

   let d2 = (total of i19) + d1
```

Once you press ESC following an ADD, UPDATE, or QUERY o the **items** table, PERFORM calculates values for the **d1** and **d2** fields and displays the values on the screen.

UPDATE

Overview

Use the UPDATE keyword to cause **PERFORM** to execute actions after the **Update** operation.

Syntax

AFTER UPDATE OF *table-list*

action
action
.
.
.
action

Explanation

AFTER UPDATE OF	are required keywords.
table-list	is a space-separated list of database tables.

Notes

1. The main use of the UPDATE keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

Examples

The following example is from the **sample** form. The action statements used in this example are described in the "Action Syntax" section of this chapter.

```
after add update query of items
    if (total of i19) <= 100 then
       let d1 = 7.50
       else
       let d1 = (total of i19) * .04

    let d2 = (total of i19) + d1
```

Once you press ESC following an **Add**, **Update**, or **Query** of the **items** table, PERFORM calculates values for the **d1** and **d2** fields and displays the values on the screen.

QUERY

Overview

Use the QUERY keyword to cause **PERFORM** to execute actions after the **Query** operation.

Syntax

AFTER QUERY OF *table-list*

action
action
.
.
.
action

Explanation

AFTER QUERY OF are required keywords.

table-list is a space-separated list of database tables.

Notes

1. The main use of the QUERY keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

Examples

The following example is from the **sample** form. The action statements used in this example are described in the "Action Syntax" section of this chapter.

```
after add update query of items
    if (total of i19) <= 100 then
        let d1 = 7.50
        else
        let d1 = (total of i19) * .04

    let d2 = (total of i19) + d1
```

Once you press ESC following an **Add**, **Update**, or **Query** of the **items** table, PERFORM calculates values for the **d1** and **d2** fields and displays the values on the screen.

REMOVE

Overview

Use the REMOVE keyword to cause **PERFORM** to execute actions before or after the **Remove** operation.

Syntax

[BEFORE | AFTER] REMOVE OF *table-list*

action
action
.
.
.
action

Explanation

BEFORE	is an optional keyword.
AFTER	is an optional keyword.
REMOVE OF	are required keywords.
table-list	is a space-separated list of database tables.

Notes

1. The main use of the AFTER REMOVE operation involves keeping track of the number of rows removed, computing statistics on the values entered into particular fields, and other bookkeeping operations.

2. Use the BEFORE REMOVE operation to cause **PERFORM** to take one or more actions before removing a row from a database table.

   For example, the following statement included a form specification prints a message on the screen whenever a user selects the **Remove** option:

```
. . .
instructions

before remove of customer
   comments reverse
   "Remember to send a notice to the sales department"
   . . .
```

   You could use the BEFORE REMOVE operation with the ABORT keyword (described later in this chapter) to prevent a user from removing the last row from a detail table.

DISPLAY

Overview

Use the DISPLAY keyword to cause **PERFORM** to execute actions after any of the **PERFORM** operations that cause data to be displayed on the screen.

Syntax

AFTER DISPLAY OF *table-list*

action
action
.
.
.
action

Explanation

AFTER DISPLAY OF	are required keywords.
table-list	is a space-separated list of database tables.

Examples

The following example is taken from the **sample** form specification file at the end of this chapter.

```
after display of orders
    let d1 = 0
    let d2 = 0
```

As soon as the data is displayed when the **orders** table is active, this control block instructs **PERFORM** to set the values in the **d1** (Ship Charge) and **d2** (Total Order Amount) fields to zero.

Action Syntax

This section provides the syntax of the following actions:

Symbol	Action
LET	Assigns values to fields
NEXTFIELD	Moves the cursor to a specific field or exits to the **PERFORM** Menu
COMMENTS	Displays a message on the Status Line
IF-THEN-ELSE	Performs other actions based upon conditions among the values in the fields
ABORT	Exits to the **PERFORM** Menu without making a change to the database

In order for these actions to compile properly, they must be included in a BEFORE or AFTER control block.

ABORT

Overview

Use the ABORT keyword in the Instructions Section of a form specification to end a current add, update, or remove action without altering the database and return to the **PERFORM** Menu.

Syntax

ABORT

Explanation

ABORT returns you to the **PERFORM** menu without executing the ADD, UPDATE, or REMOVE actions.

Notes

1. The ABORT action compares to the NEXTFIELD EXITNOW action in the following respects:

   - With NEXTFIELD EXITNOW, **PERFORM** executes an update, remove, or add, and then exits to the **PERFORM** Menu. NEXTFIELD EXITNOW is equivalent to pressing the **ESC** key.

   - With ABORT, **PERFORM** exits to the **PERFORM** Menu without executing an update, remove, or add. ABORT is equivalent to pressing the **DEL** key.

2. The ABORT keyword may be used with the EDITADD, EDITUPDATE, and REMOVE options.

   For example, suppose you maintain a master table with employee information and a detail table with information about employee projects (joined to the master table by

employee number). Projects are added and deleted on a regular basis, and you want to ensure that all employees have projects. (It is an administrative or clerical error to remove the last detail row, thereby leaving the employee with no project.) You could use the ABORT keyword with the BEFORE REMOVE operation to call a C function that checks the number of rows in the detail table. If the current row is the last detail row, the operation aborts.

LET

Overview

Use the LET action to attach a value to a field tag for display on the form.

Syntax

LET *field-tag* = *expr*

Explanation

LET is a required keyword.

field-tag is the field tag of a display-only field, of a column named in the *table/column-list*, or of a column belonging to each of the tables named in the *table/column-list*.

expr is an expression as defined below.

Notes

1. **FORMBUILD** gives an error if *field-tag* does not satisfy the above conditions.

2. You can assign values only to fields corresponding to columns in the active table or to display-only fields.

3. An expression is:

   - A field tag

   - A constant value

- One of the aggregate functions followed by the phrase OF *tagname*, where *tagname* is the field tag of a database column and not the name of a display-only field. The aggregate function values are computed over the current list.

  The aggregate functions are

COUNT	the number of rows
TOTAL	the arithmetic sum of the values of *tagname*
AVERAGE	the average of the values of *tagname* (AVERAGE can also be written as AVG)
MAX	the maximum value of *tagname*
MIN	the minimum value of *tagname*

- The keyword TODAY that returns today's date

- Any combination of the above, combined with the arithmetic operators +, −, *, and /

4. An expression can contain parentheses to make explicit the precedence of the arithmetic operators.

Examples

The following example is from the **sample** form:

```
after add update query of items
   if (total of i19) <= 100 then
      let d1 = 7.50
      else
      let d1 = (total of i19) * .04

   let d2 = (total of i19) + d1
```

Once you press ESC following an **Add**, **Update**, or **Query** of the **items** table, PERFORM calculates values for the **d1** (Ship Charge) and **d2** (Total Order Amount) fields. If the value of the **i19** (Total Price) field (all items in the order) is less than or equal to 100, then the value of the **d1** field (Ship Charge) is set to 7.50; otherwise the value is set to the sum of the **i19** (Total Price) field times .04.

The value of the **d2** (Total Order Amount) field is set to the sum of the **i19** (Total Price) field plus the value in the **d1** (Ship Charge) field.

Additional examples of the uses of the LET statement follow:

```
let f1 = f2 * 1.065

let s2 = "default string"

let f3 = (f5 + f8) * f7

let ftax = 0.065 * f_price

let f9 = average of f_price

let yr = (today - hdate)/365
```

NEXTFIELD

Overview

When using the EDITADD or the EDITUPDATE options, use the NEXTFIELD action to direct the movement of the cursor. The NEXTFIELD action overrides the default progression as determined by the ATTRIBUTES section of the form specification file.

Syntax

NEXTFIELD = \[*field-tag* | EXITNOW\]

Explanation

NEXTFIELD	is the keyword that instructs **PERFORM** to move the cursor to a particular field or to end the current **Add** or **Update**.
field-tag	is the field tag corresponding to a database column in the active table.
EXITNOW	is a keyword value for NEXTFIELD that ends the current editing.

Notes

1. You cannot change the active table by using the NEXTFIELD action to move the cursor to the field of a column in a new table.

2. The NEXTFIELD EXITNOW action compares to the ABORT action in the following respects:

- With NEXTFIELD EXITNOW, **PERFORM** executes an update, remove, or add, and then exits to the **PERFORM** Menu. NEXTFIELD EXITNOW is equivalent to pressing the **ESC** key.

- With ABORT, **PERFORM** exits to the **PERFORM** Menu without executing an update, remove, or add.

3. Since the NEXTFIELD action controls the movement of the cursor, it is effective only after the EDITADD and EDITUPDATE options.

An example from the **sample** form follows:

```
before editadd editupdate of orders
   nextfield = o20
```

In this example, as soon as you indicate that you want either the **Add** or **Update** options when **orders** is the active table, **PERFORM** is instructed to move the cursor to the **o20** column (Customer P.O.). Without this instruction, the cursor would go first to the **o11** field (Order Number), since it is the first **orders** field to appear in the ATTRIBUTES section of the form.

COMMENTS

Overview

Use the COMMENTS action to display a message on the Status Line of the screen. This contrasts with the COMMENTS attribute included in the ATTRIBUTE section that writes a message on the Comments Line.

Syntax

COMMENTS [BELL] [REVERSE] "*mstring*"

Explanation

COMMENTS	is the keyword that directs **PERFORM** to write a message on the Status Line.
BELL	is the keyword that directs **PERFORM** to ring the terminal bell as it writes the message.
REVERSE	is the keyword that directs **PERFORM** to write the message in reverse video. The default is normal video.
mstring	is the message (string) and must be enclosed in quotation marks. It must fit on one screen line and on one line of the form specification file.

Notes

1. If the REVERSE keyword is used, care must be taken on some terminals to account for the space required at the beginning of the line for the CONTROL characters.

2. The message is cleared at the next keystroke. Since **PERFORM** writes a message whenever a row is written, an **Update** or an **Add** is aborted, or a **Query** or **Remove** is made, this action is useful only for the EDITADD and EDITUPDATE keywords.

IF-THEN-ELSE

Overview

Use the IF-THEN-ELSE action to take actions conditioned upon the values in the displayed fields.

Syntax

IF *bool-expr* THEN *t-action* [ELSE *f-action*]

Explanation

IF	is a required keyword.
bool-expr	is a Boolean expression involving field tags that can take on the values true and false.
THEN	is a required keyword.
t-action	is the action or actions to be taken if *bool-expr* is true.
ELSE	is a keyword.
f-action	is the action or actions to be taken if *bool-expr* is false.

Notes

1. A Boolean expression is a combination of logical comparisons (=, < >, >, <, >=, <=) and logical operations (AND, OR, NOT) among expressions as previously defined. The operators IS NULL and IS NOT NULL may also be used.

2. For CHAR type fields only, a Boolean expression may be also of the form

   field–tag MATCHES "*string*"

   where *string* must be enclosed within quotation marks and can include wildcard characters as defined in Chapter 2, "The RDSQL Query Language."

3. *t-action* and *f-action* are either single actions as defined in this section or more than one such action between the keywords BEGIN and END.

Examples

```
if (f1 * f2 > 200) then LET f5 = -f4
                   else LET f5  = -5
```

An example from the **sample** form specification file follows:

```
after add update query of items
   if (total of i19) <= 100 then
      let d1 = 7.50
      else
      let d1 = (total of i19) * .04

   let d2 = (total of i19) + d1
```

Once you press ESC following an **Add**, **Update**, or **Query** of the **items** table, PERFORM calculates values for the **d1** (Ship Charge) and **d2** (Total Order Amount) fields. If the value of the Total Price field (all items in the order) is less than or equal to 100, then the value of the **d1** field (Ship Charge) is set to 7.50; otherwise, the value is set to the sum of the Total Price field times .04.

The value of the **d2** (Total Order Amount) field is set to the sum of the Total Price field (all items in the order) plus the value in the Ship Charge field.

The *sample* Form Specification File

The **sample** form specification file was designed for entering and maintaining data on the store's customers and orders. The first screen displays information from the **customer** table, and is labeled CUSTOMER INFORMATION. The second screen is used to enter and retrieve information about customer orders.

```
database stores

screen
{
═══════════════════════════════════════════════════════════════
═══════════════════════════════════════════════════════════════
CUSTOMER INFORMATION:

Customer Number: [c1          ]

        Company: [c4                    ]
     First Name: [c2          ]              Last Name: [c3              ]

        Address: [c5                    ]
                 [c6                    ]

           City: [c7              ] State: [c8]   Zip: [c9    ]
      Telephone: [c10               ]


═══════════════════════════════════════════════════════════════
═══════════════════════════════════════════════════════════════
}

screen

{
═══════════════════════════════════════════════════════════════
CUSTOMER NUMBER: [c1          ]          COMPANY: [c4                    ]

ORDER INFORMATION:
     Order Number: [o11          ]      Order Date: [o12           ]

     Stock Number: [i13         ]     Manufacturer: [i16]
      Description: [s14             ]                [m17             ]
             Unit: [s16             ]
                                          Quantity: [i18          ]
                                         Unitprice: [s15                ]
                                       Total Price: [i19                ]

SHIPPING INFORMATION:
 Customer P.O.: [o20         ]          Ship Charge: [d1          ]

       Backlog: [a]           Total Order Amount: [d2          ]
     Ship Date: [o21        ]
     Date Paid: [o22        ]
  Instructions: [o23                                    ]

}
end
```

```
tables
customer items stock
orders manufact

attributes
c1 = *customer.customer_num
   = orders.customer_num;
c2 = fname,
       comments = "Please enter initial if available";
c3 = lname;
c4 = company, reverse;
c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
     include = ("CA","OR","NV","WA"),
     default = "CA" ;
c9 =  zipcode, autonext;
c10 = phone, picture = "###-###-####x####";
o11 = *orders.order_num = items.order_num;
o12 = order_date, default = today, format = "mm/dd/yyyy";
i13 = items.stock_num;
    = *stock.stock_num, noentry, noupdate, queryclear;
i16 = items.manu_code, lookup m17 = manufact.manu_name
   joining *manufact.manu_code, upshift, autonext;
    = *stock.manu_code, noentry, noupdate,
          upshift, autonext, queryclear;
s14 = stock.description, noentry, noupdate;
s16 = stock.unit_descr, noentry, noupdate;
s15 = stock.unit_price, noentry, noupdate;
i18 = items.quantity, include = (1 to 50),
comments = "Acceptable values are 1 through 50" ;
i19 = items.total_price;
o20 = po_num, required,
comments = "If no P.O. Number enter name of caller" ;
a   = backlog, autonext;
o21 = ship_date, default = today, format = "mm/dd/yyyy";
o22 = paid_date, format = "mm/dd/yyyy";
o23 = ship_instruct;
d1 =  displayonly type money;
d2 =  displayonly type money;

instructions

customer master of orders;
orders master of items;
composites <items.stock_num, items.manu_code>
            *<stock.stock_num, stock.manu_code>

before editadd editupdate of orders
   nextfield = o20

before editadd editupdate of items
   nextfield = i13

after editadd editupdate of quantity
   let i19 = i18 * s15
   nextfield = o11

after add update query of items

   if (total of i19) <= 100 then
      let d1 = 7.50
      else
      let d1 = (total of i19) * .04

   let d2 = (total of i19) + d1

after display of orders

   let d1 = 0
   let d2 = 0

end
```

The CUSTOMER INFORMATION Screen

The CUSTOMER INFORMATION Screen contains fields for the entry and display of all the columns in the **customer** table. This screen could be used to add or remove a customer from the database.

Points to note about this screen:

- The **customer** table is joined with the **orders** table at the **customer_num** column. The **customer** table is the dominant table in this join.
- A comment line appears when the cursor moves into the **c2** (First Name) field.
- The **c4** (Company) field appears in reverse video.
- **PERFORM** automatically enters uppercase letters into the **c8** (State) field. The default value for the field is CA and only abbreviations for four states are allowed.
- A character pattern is specified for the **c10** (Telephone) field.

The ORDER INFORMATION Screen

The second screen contains fields drawn from the **orders**, **items**, **stock**, and **manufact** tables. This screen is used to enter information about a customer's order. Shipping information (purchase order number, instructions, date sent, and so on), and order information (order number, date, items included in the order, total prices on each item, and so on) are entered on this screen.

Points to note about the screen:

- The **c1** (CUSTOMER NUMBER) and **c4** (COMPANY) fields are repeated from the CUSTOMER INFORMATION Screen.

  The verify join between the **customer.customer_num** column and the **orders.customer_num** column prevents the assignment of an order to a nonexisting customer. When the **orders** table is active, no value can be entered into a field that does not already exist in the **customer** table.

- The **orders** and **items** tables are joined at the **order_num** column. This verify join prevents the assignment of an item to a nonexisting order number. When the **items** table is active, no value can be entered into the field that does not already exist in the **orders** table.

- The **o12** (Order Date) field has an assigned format and defaults to the current date.

- The **stock_num** column in the **items** table is joined with the **stock_num** column in the **stock** table. This is a verify join.

- The **manu_code** column in the **items** table is joined with the **manu_code** column in the **stock** table. This is a verify join.

- The **i13** (Stock Number) and **i16** (Manufacturer) fields are members of a composite join between the **items** and **stock** tables. This is a composite verify join, so no values can be entered in the **stock_num** and **manu_code** fields when the **items** table is active that do not already exist in those two columns in the **stock** table. This precludes entry of stock numbers and manufacturer codes that individually exist in the database but, as a composite, do not correspond to a unique row in the **stock** table.

To specify a unique row in the **stock** table requires both the **stock_num** and **manu_code**. For example, the **stock** table contains three rows with the stock number 1 and four rows with the manufacturer code HRO. Knowing the stock number or manufacturer code alone does not allow you to locate a unique row. You need both the stock number (1) and the manufacturer code (HRO) to specify a unique row (baseball gloves produced by Hero) in the table.

- Once the **i13** (Stock Number) and **i16** (Manufacturer) fields are filled, **PERFORM** can locate the corresponding unique row in the **stock** table. The **s14** (Description), **s16** (Unit), and **s15** (Unitprice) fields automatically display this information.

- The **i16** (Manufacturer) field is involved in a lookup join that locates the appropriate manufacturer name in the **manufact** table and places this information in the **m17** field.

- The **i18** (Quantity) field allows the entry of values 1 through 50 only, and displays a comment when the cursor moves into the field. This helps to prevent mistaken entries of an extra digit (for example, 100 in place of 10).

- The cursor does not visit the **o11** (Order Number) field when the **orders** table is the active table, because the **order_num** column in the **orders** table is a SERIAL data type.

- The entry

```
before editadd editupdate of orders
   nextfield = o20
```

  in the INSTRUCTIONS section tells **PERFORM** that when the **orders** table is the active table, the cursor first goes to the **o20** (Customer P.O.) field, rather than the **o12** (Order Date) field.

- The default value for field **o21** (Ship Date) is set to today.

- The **d1** and **d2** fields that do not correspond to any database columns. One of these is the field labeled **Ship Charge**, and is the total shipping charge for the entire order. The second is the field labeled **Total Order Amount**, and is the total charge for the entire order, including all items and the shipping charge. Each field is calculated automatically by **PERFORM**. The entry

```
after add update query of items
    if (total of i19) <= 100 then
       let d1 = 7.50
       else
       let d1 = (total of i19) * .04

    let d2 = (total of i19) + d1
```

in the INSTRUCTIONS section tells **PERFORM** that if the total of all values in the field **i19** for this order is less than or equal to 100, then set the value in the field **d1** to 7.50. If the total of all values in the field **i19** for this order is greater than 100, then set the value in field **d1** to the product of this total times .04. The Total Order Amount (the **d2** field) is the sum of all values in Total Price (the **i19** field) for the order plus the Shipping Charge (the **d1** field).

The total price of an individual item in a customer order is calculated automatically by **FORMBUILD**. This field is filled in by **PERFORM** as soon as you supply information for the fields **i13** (the Stock Number of the item), **i16** (the Manufacturer of the item), and **i18** (the Quantity of the item). **PERFORM** can do this because, in the INSTRUCTIONS section, **PERFORM** is told to calculate the value of the Total Price (the **i19** field) based upon the values entered into the **i18** (the Quantity) and **s15** (the Unit Price) fields.

- The **customer** table is the master of the **orders** table. You can easily query the **orders** table (and locate all orders placed by a customer) based upon the current row in the **customer** table by selecting the **Detail** option.

- The **orders** table is the master of the **items** table. You can easily query the **items** table (and locate all items contained in each order) based upon the current row in the **orders** table by selecting the **Detail** option.

Chapter 4

The PERFORM Screen Transaction Processor

Chapter 4 Table of Contents

Overview 5

Running PERFORM 6
Accessing PERFORM from the Main Menu 6

The PERFORM Screen 8
The Information Lines 9
The Screen Form 11
Status Lines 12
Running Operating System Commands from PERFORM 13

Entering Data 14
Data Types 14
Special Functions 16
Positioning the Cursor 16
Field Editing 17
Display Field Order 19

Data Checking 20
User Access Privileges on Multi-User Systems 21

The Current List 23

Menu Options 24
ADD 25
CURRENT 27
DETAIL 28
EXIT 30
MASTER 31
NEXT 32
OUTPUT 33
PREVIOUS 36
QUERY 37
REMOVE 42
SCREEN 44
TABLE 45
UPDATE 46

Overview

PERFORM is an **INFORMIX-SQL** program designed to streamline data entry and retrieval. After you create a screen form with **FORMBUILD**, you can use the form with **PERFORM** to query and modify the data in a database. For information about designing and building screen forms, see Chapter 3, "The FORMBUILD Transaction Form Generator."

This chapter is divided into two parts. The first part describes the following **PERFORM** operations:

- Accessing **PERFORM** from the Main Menu
- Running operating system commands while using **PERFORM**
- Using special keys to position the cursor and edit text
- Entering and editing data on the screen
- Data checking with **PERFORM**
- Controlling user privileges in **PERFORM**
- Using the current list

The second part of this chapter discusses each **PERFORM** option. The options are listed in alphabetical order.

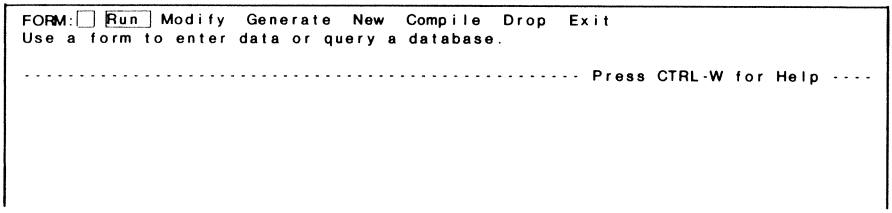
Running PERFORM

PERFORM uses the file that **FORMBUILD** generates when you compile a form specification file. This file must be in your working directory or a directory included in your DBPATH environment variable.

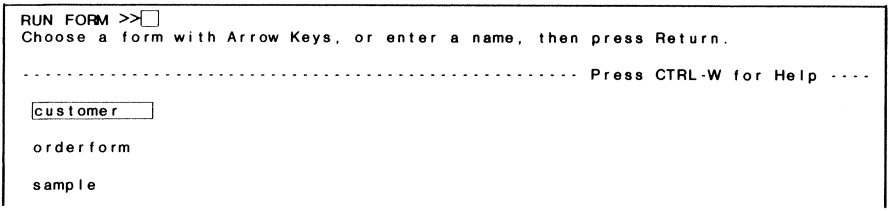
You can use **PERFORM** from the INFORMIX-SQL Main Menu or directly from the operating system. See Appendix M, "Accessing Programs from the Operating System," for information about command line usage.

Accessing PERFORM from the Main Menu

Select the **Form** option on the INFORMIX-SQL Main Menu. The FORM Menu is displayed.



Select the **Run** option on the FORM Menu. The RUN FORM Screen is displayed, with a list of available screen forms.



Type the name of the form you want to use, or use the **ARROW** keys to highlight your choice on the screen. Press **RETURN**. The form you select will appear on the screen with the **PERFORM** Menu.

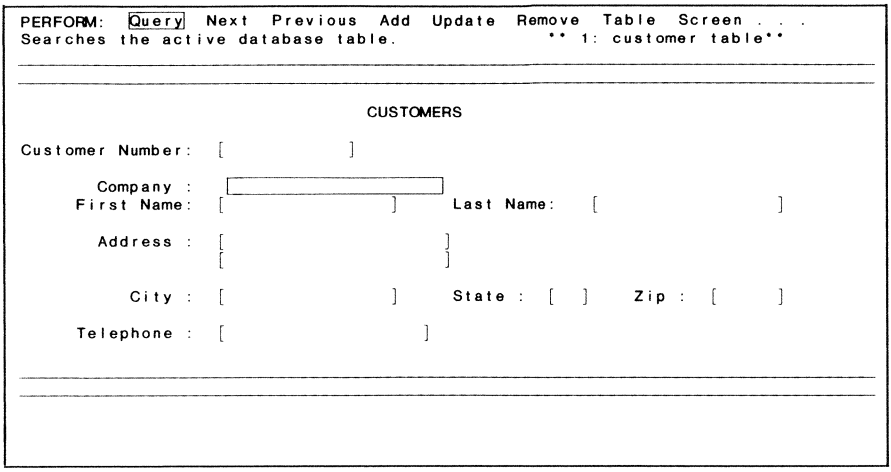


Figure 4-1. The **customer** Screen Form

The PERFORM Screen

The **PERFORM** Screen is divided into three sections:

- The first two lines of the screen (the Information Lines) display the **PERFORM** menu options, a message describing the highlighted option, and the number and name of the active table.

- The middle section of the screen (the screen form) displays the form you selected.

- The bottom two lines of the screen (the Comment Line and Status Line) display **PERFORM** messages, as well as comments specified in the form file.

The **PERFORM** menu is two pages in length. The first Information Line displays a list of menu options; the second Information Line describes the current option and indicates the number and name of the active database table. The next two screens illustrate how the two-page **PERFORM** menu appears.

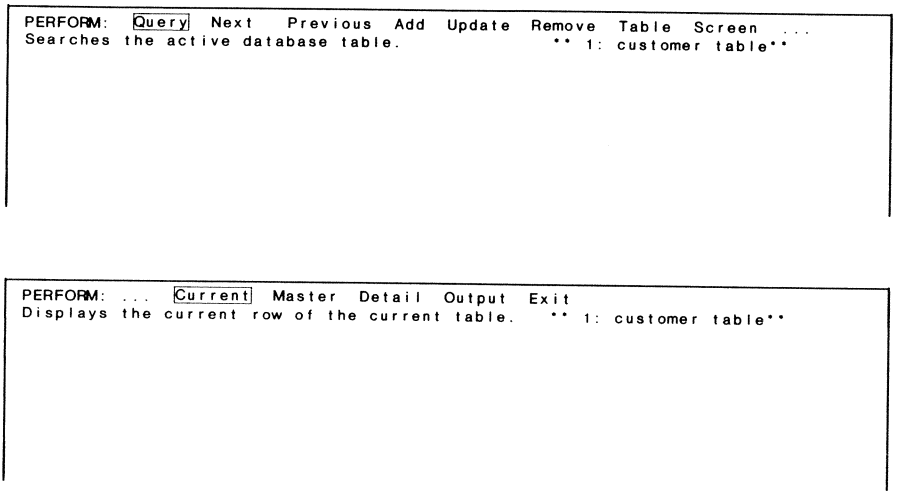


Figure 4-2. The Information Lines on a PERFORM Menu

The ellipsis on the first menu page indicates that additional menu items are available on the second menu page. The ellipsis on the second menu page indicates that additional menu items are available on the previous menu page.

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 show more options on the first 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 menu does not scroll. The highlight never rests on the ellipses: upon moving the highlight past the last or first option on each screen page, the next **PERFORM** menu page appears.

PERFORM is a menu driven program; to work with the data on the screen, you select one of the menu options. You select a menu option on the first Information Line by using the **ARROW** keys or the **SPACEBAR** to position the highlight on a menu option and then pressing return, or by typing the first letter of the menu option. **PERFORM** immediately displays the screen for the selected option. If you want to return to the menu without making any entries, press the program interrupt key. This is **DEL** on most UNIX systems or **CTRL-C** on DOS systems and some UNIX systems.

The **PERFORM** screen has thirteen menu options:

Query	Retrieves rows from the database based on search values you enter on the form and stores the rows in the current list
Next	Displays the next row in the current list
Previous	Displays the previous row in the current list
Add	Adds data to the database
Update	Modifies data in the database
Remove	Deletes a row from a database table
Table	Displays a different table in the form
Screen	Displays a different screen page of the form
Current	Restores the base current list in multi-table queries and displays the most up-to-date version of the displayed row in multiuser environments
Master	Displays the active table's master table

Detail Displays the active table's detail table

Output Writes one screen page for each selected row to an operating system file

Exit Leaves **PERFORM**

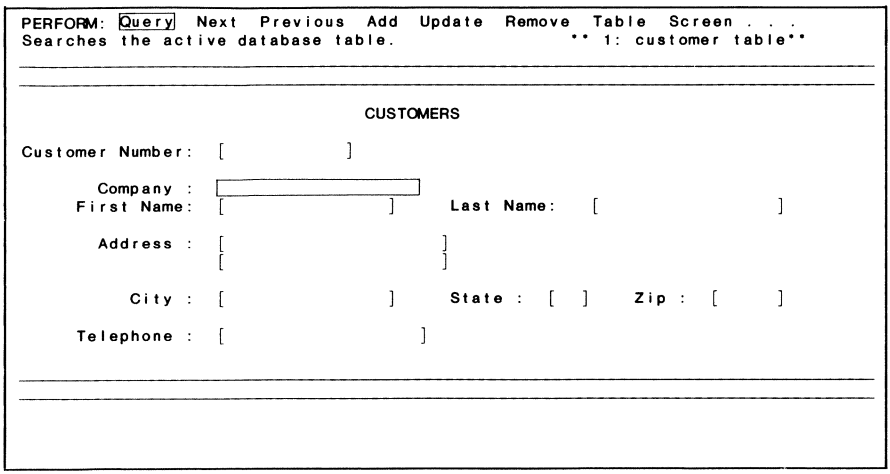
The Information Lines also indicate the number and name of the active table. Every table included in the screen form has a table number assigned according to the order in which display field tags (including joins) for the table first appear in the ATTRIBUTES section of the form specification file. This number appears next to the table name in the right-hand corner of the second Information Line when the table is active. The table number is useful for non-sequential moves to another table using the **Detail** and **Table** options.

The Screen Form

The screen form consists of one or more display fields in which **PERFORM** displays—and you enter—data. Each display field on a screen form corresponds to one or more of the database columns or to a display-only field specified in the form file. Unless you specify alternate delimiters in the form specification file, active display fields are surrounded on the screen by brackets [ ] . Fields with no delimiters are not active; values may appear in them if they are LOOKUP fields or display-only fields, but you cannot enter data into them.

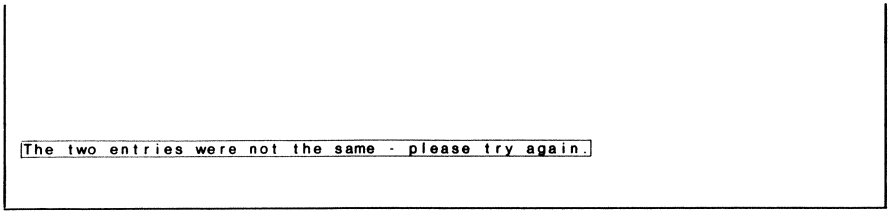
A screen form may be one or several pages in length and can contain columns from several tables. All tables included in a form must be part of the same database.

Here is how the **PERFORM** screen looks when you use the **customer** form included with the demonstration database:



Status Lines

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



Running Operating System Commands from PERFORM

To run operating system commands from the **PERFORM** Menu, press the exclamation point (!) key. **PERFORM** displays the exclamation point at the bottom of the screen. Enter an operating system command and press **RETURN**. **PERFORM** displays the results of your command and then the message Type carriage return to continue. Press **RETURN** to leave the operating system and return to **PERFORM**.

Entering Data

The **Add** and **Update** options are used to enter data directly into the database with the screen form. You must enter data of the type specified when the table was created—dates in DATE fields, money in MONEY fields, and so on. If you make a mistake in entering data, you can use the field editing keys (discussed in a following section) to correct it.

Data Types

The list below discusses the kind of data to enter for each data type. If you enter data of the wrong type, **PERFORM** displays Error in field on the Status Line. Enter an acceptable value or press the program interrupt key to cancel the option you are using. You can use the **Info** options in the RDSQL or TABLE Menu to find out the data type for each column in a table. See Chapter 9, "Database Structure and Integrity," in the *INFORMIX-SQL User Guide* for more information about data types.

CHAR	Enter letters, numbers, and symbols. During an **Add** or **Update,** the string of character data you enter may be as long as the display field.
SMALLINT	Enter whole numbers between −32,767 and 32,767.
INTEGER	Enter whole numbers between −2,147,483,647 and 2,147,483,647.
SERIAL	**PERFORM** assigns serial values automatically, so you never add data to or update a SERIAL field. However, you can enter search values in SERIAL fields when you use the **Query** option.

DECIMAL	Enter decimal numbers. The format of the number (number of places to the right and left of the decimal point) depends on the format you specified when you created the database table. If you enter a number with too many spaces after the decimal point, **PERFORM** rounds it off.
MONEY	Enter dollars-and-cents amounts without dollar signs and commas (4254.30, not $4,254.30). When you press **RETURN**, **PERFORM** automatically adds a dollar sign.
SMALLFLOAT	Enter floating-point numbers with up to seven significant digits. **PERFORM** sometimes introduces a slight discrepancy when you type numbers into SMALLFLOAT fields. The entry 1.11, for example, might display as 1.11000001 after you press **RETURN** or **ESC**. This occurs because of the way a computer stores numbers internally, and only affects you when a precision of more than seven digits is required.
FLOAT	Enter floating-point numbers with up to fourteen significant digits. The discrepancies mentioned for SMALLFLOAT data also apply to this data type if you require a precision of more than fourteen digits.
DATE	Enter dates in the form [mm]m[d]d[yy]yy, with any non-numeric characters as optional dividers (for May 2, 1985, you could enter May 2 85, 08/02/85, 8.2.85, or 08 02 1985).

Special Functions

As you enter data or a query, three special functions are available by using selected 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, and 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

You can use the following keys to position the cursor on the screen:

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 (and 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 (and 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, you may find it 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. For example, 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. For example, you could use insert mode to add an *i* to *Rchard*.

Whenever the cursor enters a field, PERFORM 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. Move the cursor into a new field and you are automatically in typeover mode.

Edit data that appears in a field with 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 (and Ins on DOS systems) shifts between insert and typeover mode. When you access PERFORM, you will be in typeover mode.
Delete Forward	The CTRL-D key (and F9 on DOS systems) deletes everything from the current cursor position to the end of the field.
Repeat Data	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 the field. On DOS systems the F3 key enters the most recently displayed value in a field.

Clear Screen The CTRL-C key (and F10 on DOS systems) clears any search criteria you have entered with the **Query** option.

Display Field Order

The cursor ordinarily moves through the display fields in the order in which their field tags are listed in the ATTRIBUTES section of the form file. This order may be modified by a NEXTFIELD statement in the INSTRUCTIONS Section of the form file.

Data Checking

The attributes and instructions in the form file can affect data entry, data storage, data display, and cursor movement when you use the **Add**, **Update**, and **Query** options. If you get undesired displays or cursor movement, you may want to modify the form file. The effects of some attributes and instructions are listed below, followed by the relevant options. (See Chapter 3, "The FORMBUILD Transaction Form Generator," for details about attributes and instructions.)

- The case of character data on the screen is different than what you type; check for UPSHIFT and DOWNSHIFT attributes (**Add**, **Update**, **Query**).

- SMALLFLOAT or FLOAT data on the screen is different from what you type; check for a FORMAT attribute specifying the number of places to the right and left of the decimal point causing rounding off (**Add**, **Update**).

- **PERFORM** displays This value is not among the valid possibilities; check for an INCLUDE attribute specifying acceptable values and ranges of values (**Add**, **Update**).

- The terminal beeps and does not echo your entry on the screen; check for a PICTURE attribute limiting data entry to a specified pattern of variables and literals (**Add**, **Update**).

- The cursor skips over a bracketed display field; if the field is not a SERIAL field, check for a NOENTRY attribute (for **Add** only), a NOUPDATE attribute (for **Update** only), or a NEXTFIELD action (both **Add** and **Update**).

- **PERFORM** displays This field requires an entered value; check for a REQUIRED attribute (**Add**). You must explicitly enter all values for a field with the REQUIRED attribute unless you have specified a value with DEFAULT.

- **PERFORM** displays Please type again for verification; check for a VERIFY attribute (**Add**, **Update**).

- Data you enter appears on the screen justified to the right; check for a RIGHT attribute (**Add**).

- Numeric data appears on the screen justified to the right and padded with leading zeros; check for a ZEROFILL attribute (**Add**).

- A value you did not enter appears in a field; check for a DEFAULT attribute, a PICTURE attribute with literals (**Add, Query**), a joined field (**Add**, sometimes **Query**), or a LET action (**Add**).

- The cursor moves automatically to the next field after this field is full; check for an AUTONEXT attribute (**Add, Update**).

- A line of text appears on the screen; check for a COMMENTS attribute (**Add, Update, Query**).

- A line of text appears on the screen, in regular or reverse video, and/or the terminal bell rings; check for a COMMENTS action (**Add, Update, Query**).

- Data stores automatically before you press ESC; check for a NEXTFIELD EXITNOW action (**Add, Update**).

- PERFORM displays This is an invalid value--it does not exist in tablename. Check for a verify join (**Add, Update**).

User Access Privileges on Multi-User Systems

Access privileges controlled by the GRANT and REVOKE statements can affect your ability to display, enter, modify, and remove data for a table or a display field. A message like

```
Permission not granted to allow update
```

means that you do not have access privileges. Use the **Info** option on the RDSQL or TABLE Menu to find out the access privileges for a particular table.

See Chapter 2, "The RDSQL Query Language," for more information about privileges.

The Current List

The current list is a temporary storage area where PERFORM stores the results of a query. It can hold from one row to all the rows in a database. Whenever you select the **Query** option, PERFORM erases the existing current list to make room for new query results.

The **Query**, **Next**, **Previous**, **Remove**, and **Update** options all involve the current list. The **Query** option finds all rows that satisfy the search conditions and puts them in the current list. The **Next** and **Previous** options step through the rows in the current list in sequential order. The **Update** and **Remove** options can only work with rows in the current list.

Menu Options

The menu options you can use with **PERFORM** are described in detail on the following pages. They are listed in alphabetical order, rather than menu order, for easy reference.

ADD

Overview

Use the **Add** option to create new rows in the active table. You can type data on a screen form, review it, edit it, and store it in the database.

Procedure

1. Type a to select the **Add** option. **PERFORM** clears all data from the screen form except joined fields and some display-only fields, displays spaces or default values in the fields, and positions the cursor in the first field.

2. Fill in the form with the values you want to enter. If you enter data inappropriate to the data type of the display field, **PERFORM** displays Error in field. You cannot move on to the next display field until you correct the entry.

3. Press **ESC** to store the row, or the program interrupt key to cancel the addition and redisplay the **PERFORM** menu options.

Notes

1. If **PERFORM** displays

   ```
   Could not insert new row - duplicate value in a unique
   index column
   ```

   it does not store a row when you press **ESC**. You are trying to enter a duplicate value where it is not permitted. Use the **Info** option on the RDSQL or TABLE Menu (or execute an **RDSQL** INFO statement) to check for unique indexes.

2. In a multi-user environment, tables may be fully or partly unavailable to you because another user has invoked the LOCK TABLE statement or because another user is updating a row that you attempt to update or remove. In such a case, **PERFORM** displays an error message.

3. **PERFORM** sometimes introduces a slight discrepancy when you enter numbers in SMALLFLOAT or FLOAT fields. The entry 1.11, for example, might display as 1.11000001 after you press **RETURN** or **ESC**. This occurs because of the way a computer stores numbers internally.

Related Options

Update, Remove

CURRENT

Overview

The **Current** option rereads and redisplays the current row in the current list for the active table.

Procedure

1. Type c to select the **Current** option.

2. PERFORM displays the most up-to-date version of the screen you were looking at before you moved to another table.

Notes

The **Current** option is useful in two situations:

1. In a multi-user environment another user may modify the information corresponding to a display field on your screen. When you use the **Current** option, PERFORM rereads the row, displaying the most recent information.

2. When a form includes a join field, each table represented on the screen form has its own current list. Looking at the information in the active table may make you "lose your place" in one or more of the other current lists. The **Current** option returns you to your original position in the active table's current list.

DETAIL

Overview

The INSTRUCTIONS Section of the form file may include one or more master-detail table relationships for tables with join fields to simplify multi-table queries. The **Detail** option automatically selects, displays, and queries the active table's detail table.

Procedure

1. Enter d to select the **Detail** option.

2. If the active table has no detail table, PERFORM displays an error message. If the active table has one or more detail tables specified, PERFORM locates and displays the active table's first detail table, no matter what the detail table's absolute table number is.

3. PERFORM automatically runs a query on the detail table, using the current values in the fields in the master table that join the fields in the detail table as search values, and puts the rows that satisfy the query conditions in the detail table's current list. You can use the **Next** and **Previous** options to examine the rows. If no rows are found, PERFORM displays

   ```
   There are no rows satisfying the conditions.
   ```

4. Type m to make the master table the active table again. You will see the first master table row with join field values matching the current values in the detail table.

Notes

1. 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.

2. If more than one detail table has been specified for a master table in the INSTRUCTIONS Section, typing **d** displays and queries the first; typing **d** preceded by the number of another detail table displays and queries the other.

3. You use a table number to query any *detail* table that joins the active table, even if no master-detail relationship is specified. Type **4d**; if table number 4 joins the active table, **PERFORM** queries table 4 and it becomes the new active table. However, **PERFORM** displays an error message when you type **d** without a table number if no master-detail relationship has been specified in the INSTRUCTIONS Section.

Related Options

Master

EXIT

Overview

Use the **Exit** option to exit from **PERFORM**.

Procedure

1. Type **e** to run the **Exit** option.

2. **PERFORM** returns you to your starting place, either the FORM Menu or the operating system.

MASTER

Overview

Use the **Master** option to move directly from a detail table to its master table.

Procedure

1. Type m to select the **Master** option.

2. If the active table has no master table, PERFORM displays an error message. If the active table has a master table, PERFORM displays the master table with the row found joining the detail table.

3. You must declare a **Master-Detail** relationship in the INSTRUCTIONS Section in order to use this option.

Related Options

Detail

NEXT

Overview

Use the **Next** option to step forward through the rows in the current list.

Procedure

1. Use the **Query** option to put the rows you want to inspect in the current list.

2. Type n to run the **Next** option. PERFORM displays the next sequential row in the current list.

3. Type n repeatedly. When you reach the last row in the current list, PERFORM displays

   There are no more rows in the direction you are going.

Notes

1. If you want to move forward several rows at once, enter a number before the **Next** option; for example, entering 10n skips ahead ten rows.

Related Options

Query, Previous

OUTPUT

Overview

Use the **Output** option to write the displayed screen page (one row or all rows in the current list) to a new or existing operating system file. The screens in the operating system file will look exactly the same as they do on the PERFORM Screen, including data, display field labels, boxes, lines, and so on. You can use the operating system utilities to edit and print the file.

Procedure

1. Use the **Query** option to create a current list containing the row or rows you want to print. If you want to output only one screen page, use the **Next** and **Previous** options to display that page.

2. Type o to select the **Output** option. In the upper left corner of the screen, PERFORM displays

   Enter output file (default is perform.out):

3. Enter the name of the file in which you want PERFORM to place the output and then press RETURN, or simply press RETURN to select the default file name given. If you enter a name for the file, it becomes the default name for this session with PERFORM and appears with subsequent selections of the **Output** option.

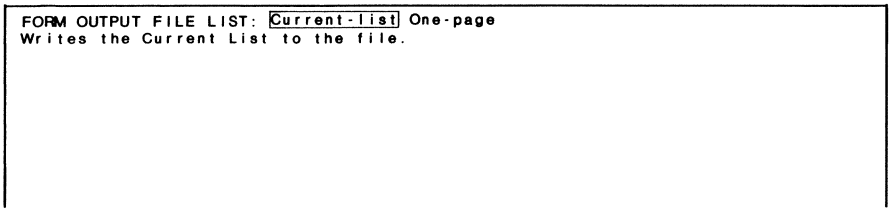
4. Then PERFORM asks

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

Type a or press **RETURN** to append the information to the file you selected in Step 3. Type c to create a new file with the name you indicated in Step 3.

If you enter an existing filename in Step 3 and select the **Create** option in Step 4, **PERFORM** overwrites the old version of the file when you run the **Output** option. You lose any data that was stored in the file.

5. Now **PERFORM** asks:



Type c or press **RETURN** to write a page for every row in the current list. Type o to write only the page that currently appears on the screen.

6. **PERFORM** writes the rows you selected to the file you indicated, displaying a counter at the bottom of the screen:

```
OUTPUT ROW NUMBER 1
```

If you selected the **Current-list** option in Step 5, **PERFORM** displays each page that it writes and updates the counter as it does so.

Notes

1. The **Output** option copies one page of a screen form for each row in the current list. To copy more than one page, use the **Output** option separately with each page. If you want to copy all of the pages of a three-page form, for example, use the **Output** option to copy all of the first pages in the current list, then use the **Screen** option to display the second page, run the **Output** option again, use the **Screen** option to display the third page, and run the **Output** option once more. If there is more than one row in the current list, all the first pages will be output together, followed by all the second pages, and so on.

PREVIOUS

Overview

Use the **Previous** option to display prior rows in the current list.

Procedure

1. Use the **Query** option to put the rows you want to look at in the current list.

2. Type n to display the next row.

3. Type p to use the **Previous** option. PERFORM displays the previous row (in this case, the first row) in the current list.

4. When you reach the first row in the current list, type p. PERFORM displays

   There are no more rows in the direction you are going.

5. You can use the **Previous** option whenever you want to display prior rows in the current list.

Notes

1. If you want to move backward several rows at once, enter a number before the **Previous** option; for example, entering 10p skips back ten rows.

Related Options

Query, Next

QUERY

Overview

Use the **Query** option to search for database rows and columns with specified values based on search values you enter directly into the display fields on a screen form. You can specify the search criteria with ten different query operators, including six relational operators, a range operator, a wildcard operator, and highest/lowest value operators. PERFORM finds all the database rows that satisfy the conditions and puts them in the current list. You can use the **Next** and **Previous** options to view them.

Procedure

1. Type q to select the **Query** option. PERFORM clears the fields in the active table of all data (except data in joined fields with no QUERYCLEAR attribute) and puts the cursor in the first field.

2. Enter search values in one or more display fields using the syntax described in the next section. To find all of the rows in the table, do not enter any search values.

   If a display field is too short to hold the search value you enter, PERFORM creates a work space at the bottom of the screen. When you press RETURN, PERFORM removes the work space. The display field contains what you entered in the work space even though you can only see the part of it that fits in the field.

3. Press ESC to run the query; press the program interrupt key to cancel the query and display the PERFORM Menu again. When you run the query, PERFORM searches the active table, puts all the rows that satisfy the conditions in the current list, and displays the first matching row on the screen. A Status Line message reads

   ```
   # row(s) found
   ```

where # is the number of rows containing the specified search value(s) in the specified display field(s). You can use the **Next** and **Previous** options to look at the rows in the current list. If no rows satisfy the conditions, the status line message reads

There are no rows satisfying the conditions

Syntax

Query Operator Syntax

Operator	Name	Data Types	Patt
=	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
|	Or	all	x|y
:	Range	all	x:y
?	Wildcard (n=1)	CHAR	?x, x
*	Wildcard (n≥0)	CHAR	*x, x
>>	Highest value	all	>>
<<	Lowest value	all	<<

Explanation

=	The default query operator; if you do not enter another operator, **PERFORM** assumes the equal sign. Enter the equal sign by itself to search for a database row that contains a null CHAR column; enter =* to find a row that contains a column with an asterisk.
x	Any search value with the appropriate data type for the search field. Enter the search value immediately after any one of the first six query operators in the table above. Do not leave a space between the query operator and the search value.
>	For CHAR data, *greater than* means later in ASCII ordering (a>A>1). For DATE data, *greater than* means after. (Appendix F is an ASCII chart.)
<	For CHAR data, *less than* means earlier in ASCII ordering (1<A<a). For DATE data, *less than* means before.
y	Any search value with a value higher than **x**.
|	The search operator that signifies *or*. 110|118|112 in the Customer Number field, for instance, would search for rows with the value 110, 118, or 112 in the *customer_num* column.
:	The search operator that specifies a range. You must give the lower value before the search operator and the higher value after the operator in a range query. Queries with the range operator are inclusive. The search criterion 1:10 would find all rows with a value in that column from one through ten, inclusive.
?	A wildcard character. It represents a single character. A ?ick search value in the First Name display field of the *orderform* form would find "Dick," "Rick," "Nick," and so on.

* A wildcard character. It represents zero or more characters. An S* search value in the Last Name display field of the **orderform** form would find Sadler and Sipes. An *r search value would find five names: Baxter, Jaeger, Miller, Sadler, and Vector.

>> The highest-value search operator. Enter it (with no search value) in a display field to find the highest value for the field.

<< The lowest-value search operator. Enter it (with no search value) in a display field to find the lowest value for the field.

Notes

1. Because of the way a computer stores floating-point numbers, you may not be able to retrieve FLOAT and SMALLFLOAT data by querying for the exact value you entered. You can solve this problem by using a range query. Specifying a FORMAT with a few places to the right of the decimal point in the FORMBUILD Attributes Section may also help.

2. If the RIGHT attribute is specified for a display field, you may have to use an asterisk in front of a search value. (RIGHT does not right-justify the search value after you enter it.)

3. Although the literals in PICTURE specifications appear on the screen when you add and update data, they do not appear on the screen when you query. If you enter the wrong literal value, your search will not be successful. A COMMENTS entry in the form file can help you avoid this problem.

Related Options

Next, Previous

REMOVE

Overview

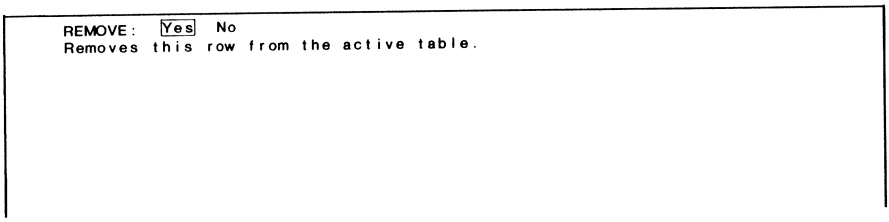
Use the **Remove** option to delete the row on the screen from the active table.

Procedure

1. Use the **Query**, **Next**, and **Previous** options to display the row you want to delete.

2. Type r to select the **Remove** option.

3. **PERFORM** displays



Enter y to delete the row, or n to keep it. In either case, the **PERFORM** Menu appears on the screen next. The message

```
Row deleted
```

appears at the bottom of the screen when you remove a row.

Notes

1. You cannot remove a verify join row from one table (generally the master table, against which the join field is verified) unless you first remove all the rows that join it in other tables (generally detail tables, which are verified against the master table). For example, using the **orderform** form, you can remove rows in the **items** table. However, you cannot remove a row in the **orders** table without removing all **items** rows that join that row because the Order Number display field is a verify join.

Related Options

Add, Update

SCREEN

Overview

Use the **Screen** option to cycle through the screen pages of the form.

Procedure

1. Type **s** to run the **Screen** option.

2. **PERFORM** displays the next screen page of the form. When you reach the last screen page, type **s** to display the first page again.

TABLE

Overview

Use the **Table** option when there is more than one table in the screen form and you want to select a new active table. Each table has a table number assigned according to the order in which display field tags (including joins) for the table first appear in the ATTRIBUTES section of the form file. This number appears next to the table name in the screen Information Lines when the table is active. The **Table** option steps through the tables in table-number order starting with the active table.

Procedure

1. Type t. If your screen form includes fields from two or more tables, **PERFORM** automatically selects and displays whichever page of the screen form contains the greatest number of fields from the new active table and surrounds those fields with delimiters; fields belonging to other tables do not have brackets.

   If each table is on a separate screen page, **PERFORM** displays the screen page for the new active table.

2. Type t again. **PERFORM** displays the next sequential table. When you reach the last table, **PERFORM** displays the first table again.

Notes

1. If you know the number of the table you want to view next, you can go directly to that table without passing through the intervening tables. For example, suppose there are five tables in your form and you are looking at table number 4. If you want to see table number 2 next, type 2t and **PERFORM** displays table number 2. Tables number 5 and 1 are skipped.

UPDATE

Overview

Use the **Update** option to modify the data in the displayed row of the current list.

Procedure

1. Use the **Query**, **Next**, and **Previous** options to display the row you want to modify.

2. Type u to run the **Update** option. PERFORM puts the cursor in the first active field.

3. Edit the data, modifying as many display fields as you like.

4. Press ESC to store the changed row or the program interrupt key to ignore the changes and display the menu again.

Notes

1. You cannot update a field that is a verify join field for another table without first updating the relevant field in the other table.

2. If you press ESC after selecting the **Update** option, PERFORM displays

   ```
   This row has been changed
   ```

   whether or not you actually changed anything.

3. PERFORM sometimes introduces a slight discrepancy when you enter numbers in SMALLFLOAT or FLOAT fields. The entry 1.11, for example, might display as 1.11000001 after you press RETURN or ESC. This occurs because of the way a computer stores numbers internally.

Chapter 5

The ACE Report Writer

Chapter 5 Table of Contents

Overview 5

Creating and Compiling a Custom Report 5
Using the Menus to Create a Report 5
Generating a Default Report 6
Creating a Custom Report 7
Creating a Report from the Command Line 8
Command Line Options 9

Information About ACE 11
ACE Filename Conventions 11
Using Expressions in a Report Specification 12
ACE Error Messages 14

Demonstration Database Sample Reports 15

Structure of a Report Specification File 16

DATABASE Section 17

DEFINE Section 18
PARAM 19
VARIABLE 21

INPUT Section 24
PROMPT FOR 25

OUTPUT Section 27
REPORT TO 28
LEFT MARGIN 30
RIGHT MARGIN 31
TOP MARGIN 33
BOTTOM MARGIN 34
PAGE LENGTH 35

SELECT Section 37

FORMAT Section 41

EVERY ROW 43
Control Blocks 47
AFTER GROUP OF 49
BEFORE GROUP OF 52
FIRST PAGE HEADER 55
ON EVERY ROW 57
ON LAST ROW 59
PAGE HEADER 61
PAGE TRAILER 63
Statements 65
FOR 66
IF THEN ELSE 68
LET 70
NEED 72
PAUSE 73
PRINT 75
PRINT FILE 78
SKIP 79
SKIP TO TOP OF PAGE 80
WHILE 82
Aggregates 83
ASCII 86
CLIPPED 88
COLUMN 90
DATE 92
DATE() 93
DAY() 94
LINENO 95
MDY() 96
MONTH() 97
PAGENO 98
SPACES 99
TIME 100
TODAY 101
USING 102
WEEKDAY( ) 110
YEAR( ) 111

Overview

ACE is a general-purpose relational report writer that produces reports based on the tables of a database. ACE can draw information from several tables based on relationships among the tables specified at the time you design the report.

Creating and Compiling a Custom Report

You can create a report specification file in one of two ways: You can use the **Report** option on the INFORMIX-SQL Main Menu, or work directly with the appropriate programs from the operating system command line. Either alternative requires that you have already created the database and all the tables from which the report will draw information. The following two subsections describe these alternative procedures. They do not, however, describe the rules for constructing or modifying the report specification file. These rules are defined in the remaining sections of this chapter.

Creating a report from the command line is also described in Appendix M, "Accessing Programs from the Operating System."

Using the Menus to Create a Report

The procedure for creating, compiling, and running a report from the REPORT Menu is described in the next two sections. The first section explains the procedure used to produce a default report. The second section covers the steps involved in the production of a custom report. A more detailed description of each procedure is presented in Chapter 8, "Creating and Printing Reports," of the *INFORMIX-SQL User Guide.*

Generating a Default Report

To create a default report using the INFORMIX-SQL menu system, follow these steps:

1. Select the **Report** option on the INFORMIX-SQL Main Menu and then the **Generate** option on the REPORT Menu.

2. If there is no current database, the CHOOSE DATABASE Screen appears. After you select a database, the GENERATE REPORT Screen is displayed. Enter the name you want to assign to the report (for example **newreport**). (Do not use the **.ace** filename extension; INFORMIX-SQL automatically adds the required extension.)

3. INFORMIX-SQL prompts you for the name of the table you want it to use to create the default report. Once you enter the table name, INFORMIX-SQL automatically compiles the report specification and displays the REPORT Menu. The report is now available to be used.

4. Select the **Run** option on the REPORT Menu to run the report.

The default report specification file formats a report as a list of all columns in the table included in the report. It does not provide any special instructions to ACE about how to display the data, nor does it include instructions to perform data manipulations. Only one table contributes information to a default report.

Creating a Custom Report

To create a customized report using the INFORMIX-SQL Menu system, follow these steps:

1. Complete the steps described in the previous section, "Generating a Default Report."

2. The REPORT Menu should now appear on the screen. Select the **Modify** option on the REPORT Menu.

3. The MODIFY REPORT Screen appears. Enter the name of the default report (**newreport**) just created.

4. If you have not specified an editor previously in the session or set the DBEDIT environment variable as described in Appendix B, INFORMIX-SQL asks you to select the editor with which you want to work. Press RETURN if you want to select the editor whose name is displayed on the top line of the screen. If you want to work with a different editor, enter the name of the editor and then press RETURN. INFORMIX-SQL calls the editor with the default report specification file.

   Modify the specification to include the data you need and the appearance you desire. Exit from the editor.

5. The MODIFY REPORT Menu is displayed. Select the **Compile** option.

6. If your report specification file compiles correctly, a message to that effect is displayed, and ACE creates a report file with the filename extension **.arc** (for example, **newreport.arc**). Go to Step 8. If your report specification file contains errors, a message to that effect is displayed, and ACE creates a report file with the filename extension **.err** (for example, **newreport.err**). Go to Step 7.

7. Select the **Correct** option from the COMPILE REPORT Menu. INFORMIX-SQL calls your system editor with the report specification file with the compiler errors. When correcting the errors, you need not delete the error messages. INFORMIX-SQL does that for you. Return to Step 5.

8. When the compilation is successful, select the **Save-and-exit** option on the MODIFY REPORT Menu. The REPORT Menu is displayed. The report is now available for use.

9. Select the **Run** option on the REPORT Menu and run the report.

As an alternative to using the **Generate** option and creating a default report specification, you can select the **New** option. INFORMIX-SQL calls your system editor, and you enter all of the report specification instructions.

Creating a Report from the Command Line

To create a customized report specification directly from the operating system command line, follow these steps:

1. Use the system editor to create a report specification file. Append the extension **.ace** to the filename.

2. Compile the specification with the ACEPREP program. Call ACEPREP as **saceprep**. You can omit the **.ace** filename extension when you call ACEPREP.

   For example, this is the command line used to compile the **newreport.ace** specification file:

   ```
   saceprep newreport
   ```

3. If the compilation is successful, ACE creates a compiled report file called **newreport.arc** and you are finished creating your customized report. Go to Step 5. If errors are detected in the report specification, a **newreport.err** file is created. Go to Step 4.

4. Use the system editor to edit this specification. Remove all error comments from the specification file. Overwrite the file **newreport.ace** with this corrected version. Go to Step 2.

5. To run the **newreport.arc** report, use the ACEGO program. Call ACEGO as **sacego**. Do not include the **.arc** filename extension when you call ACEGO.

   For example, this is the command line used to run the **newreport.arc** report:

   ```
   sacego newreport
   ```

Creating a report from the command line is also described in Appendix M, "Accessing Programs from the Operating System."

Command Line Options

Three command line options are available for use with ACE.

-o Use the **-o** (output) option, followed by the pathname of a directory, to indicate the directory where ACEPREP places its output file. If you do not use this option, ACEPREP puts the file in your working (current) directory.

For example, to instruct ACEPREP to place the output file from compiling the **newreport** specification in the **output** directory, use the following command line:

```
saceprep -o output newreport
```

-s Use the **-s** (silent) option with both ACEGO and ACEPREP to suppress all nonessential screen messages. For example, use this command line to suppress program banners in ACEPREP:

```
saceprep -s newreport
```

-d Use the **-d** (database) option, followed by the name of a database, to override the database that is named in the report specification. For example, to substitute the **sales** database for the database included in the **newreport** specification, use this command line:

```
sacego -d sales newreport
```

The report specification file, the compiled report specification, and the database files used in the report must be in your working directory or in a directory named in the DBPATH environment variable.

ACE Filename Conventions

ACE uses the following file-naming conventions.

- A report specification filename can be up to eight (on DOS systems) or ten (on UNIX systems) characters long. The filename must have an **.ace** filename extension. Without the **.ace** filename extension the file is not recognized by the ACE compiler. On UNIX systems the compiler differentiates between uppercase and lowercase letters in filenames. No such distinction is made on DOS systems.

  When you use the **New** or **Generate** options on the REPORT Menu, INFORMIX-SQL automatically adds the **.ace** extension to the filename; you must not include it in the filename you choose.

- When you compile a report specification, you can omit the filename extension if you like.

- The extension that the ACE compiler gives the output file depends upon whether the compile is successful. If the compilation is successful, the extension **.arc** is appended to the filename. If the compilation is unsuccessful, the extension **.err** is appended to the filename.

  An **.err** file is a text file that contains the original specification file along with error messages describing and pointing to the problem ACE found.

Using Expressions in a Report Specification

An expression can be anything from a simple numeric or alphabetic constant to a more complex series of column values, functions, quoted strings, operators, and keywords. ACE evaluates expressions when it generates a report. It can display the result of the evaluation, assign it to a variable, or use it in a calculation.

When ACE evaluates an expression, it combines elements of the expressions that are separated from each other by operators. It combines elements in the order shown in Figure 5-1. You can use parentheses to override this order.

Operator	Function	Precedence
–	Unary minus	1
**	Exponentiation	2
*	Multiplication	3
/	Division	3
+	Addition	4
–	Subtraction	4
IS [NOT] NULL	Presence/absence of a value	5
matches	Equality for strings	5
=	Equal	5
!= or <>	Not equal	5
>	Greater than	5
<	Less than	5
>=	Greater than or equal	5
<=	Less than or equal	5
not	Not	6
and	And	7
or	Or	8

Precedence of Operators: 1 is highest, 8 is lowest.

Figure 5-1. Operator Precedence

When this manual refers to a numeric expression (*num-expr*), you can supply any type of expression, including character, as long as ACE can evaluate it as a number. The character string "123" is a valid numeric expression, while "m23" is not.

Similarly, *date-expr* is an expression that ACE can evaluate as a date. You can use a quoted string ("01012010" or "1-1-2010") or an INTEGER that evaluates to a legal date.

A quoted string is any string of characters in quotation marks. You can use a quoted string anywhere ACE requires a type CHAR expression.

Exponents are treated as integers and not as decimals. If a decimal is provided as an exponent, ACE truncates the number

to an integer. For example, the expression `4 ** 3.4` is truncated to `4 ** 3` before it is evaluated.

A unary minus indicates or changes the algebraic sign of a value (from positive to negative or from negative to positive). It operates on a single operand.

ACE Error Messages

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

Demonstration Database Sample Reports

The sample reports in this chapter are taken from the following list. You may see additional reports included with the database which are available for further study. These reports illustrate a wide variety of the commands available with ACE.

mail1.ace	A simple report that generates mailing labels
mail2.ace	A more sophisticated report that produces one column of mailing labels
mail3.ace	An interactive report that generates one to three columns of mailing labels
clist1.ace	A report that lists customer information
clist2.ace	An interactive customer report
ord1.ace	A custom report of orders placed with the store
ord2.ace	A second customer order report
ord3.ace	An interactive report that lists daily orders

Appendix E contains the full text of each sample report specification.

Structure of a Report Specification File

A report specification file contains the instructions that specify what data a report will include and how that data appears. A report specification consists of three required sections (DATABASE, SELECT, and FORMAT) and three optional sections (DEFINE, INPUT, and OUTPUT). You must observe the order of the sections when you write an ACE report specification.

- **DATABASE Section:** Each report specification must begin with a DATABASE section identifying the database you want the report to use.

- **DEFINE Section:** The optional DEFINE section is used to declare variables used in the report and parameters the report can accept from the command line.

- **INPUT Section:** The INPUT section is optional and is used to pass parameters to the report.

- **OUTPUT Section:** The optional OUTPUT section is used to control page length and margin width, and to direct the output from the report to a file, system printer, or (on UNIX systems) a pipe.

- **SELECT Section:** The SELECT section specifies the columns and tables the report is based upon.

- **FORMAT Section:** The FORMAT section appears next and includes commands that determine how the data in the report appears.

You can include comments anywhere in an ACE report specification by enclosing the comments within a set of { } braces.

DATABASE Section

Every report specification must have a DATABASE section. The DATABASE section specifies the database ACE uses as the basis of the report. You can override the database that you specify in this section with the **-d** command line option. See the section "Command Line Options" for more information.

The DATABASE section must be the first section in an ACE report specification. It consists of the DATABASE keyword, followed by the name of the database, and ends with the END keyword. The structure of the DATABASE section is shown below.

```
DATABASE
      database-name
END
```

Examples

The following example DATABASE section is from the **clist1.ace** report:

```
database      {use stores database}
   stores
end
```

DEFINE Section

An ACE report specification can optionally contain a DEFINE section. The DEFINE section is used to declare variables used in the report and parameters the report can accept from the command line.

The DEFINE section begins with the DEFINE keyword and ends with the corresponding END keyword. The variable definition list appears between these keywords and is composed of one or more PARAM and/or VARIABLE statements. The structure of the DEFINE section is shown below.

```
DEFINE
    PARAM statement(s)
    VARIABLE statement(s)
END
```

The PARAM and VARIABLE statements are described in the next two sections.

PARAM

Overview

This statement allows you to use arguments specified on the command line at the time you run the report in an ACE report. It declares a variable whose initial value is that of a command line argument. To use PARAM, you must call ACE from a custom user menu (see Chapter 6, "User-menu") or the command line (see the section "Compiling and Running a Report," earlier in this chapter).

Syntax

PARAM[*int1*] *var-name*
{CHAR(*int2*) | DATE | DECIMAL(*int3,int4*) | FLOAT |
INTEGER | MONEY | SMALLFLOAT | SMALLINT }

(You must include the square brackets in the above syntax; they do not indicate an optional parameter.)

Explanation

PARAM	is a required keyword.
int1	is a required integer that specifies the position of the argument on the command line. The first argument is number 1.
var-name	is the name of the variable that you are declaring—it will initially have the value of a command line argument when you run the report.
CHAR	is a keyword that declares the variable as type CHAR.
int2	is an integer that specifies the length in characters of a type CHAR variable.

DATE	is a keyword that declares the variable as type DATE.
DECIMAL	is a keyword that declares the variable as type DECIMAL.
int3,int4	are two integers that describe a type DECIMAL variable. These integers are not used by ACE and are accepted only to provide compatibility with RDSQL. You can omit them without any effect.
FLOAT	is a keyword that declares the variable as type FLOAT.
INTEGER	is a keyword that declares the variable as type INTEGER.
MONEY	is a keyword that declares the variable as type MONEY.
SMALLFLOAT	is a keyword that declares the variable as type SMALLFLOAT.
SMALLINT	is a keyword that declares the variable as type SMALLINT.

Notes

1. You can define a total of 100 variables using PARAM and VARIABLE statements in an ACE report specification.

2. If a report specification uses a PARAM statement and you fail to provide arguments on the command line when you run the report, ACE gives an error message.

3. If you want to use a variable defined by a PARAM statement in the SELECT section, you must precede the variable name with a dollar sign. Refer to the "SELECT Section" later in this chapter for more information.

4. See "Database Data Types" in Chapter 2, "Using RDSQL," for a complete discussion of variable data types.

VARIABLE

Overview

This statement declares a variable that you can use in an ACE report specification.

Syntax

VARIABLE *var-name*
{CHAR(*int2*) | DATE | DECIMAL(*int3,int4*) | FLOAT |
INTEGER | MONEY | SMALLFLOAT | SMALLINT }

Explanation

VARIABLE	is a required keyword.
var-name	is the name of the variable that you are defining.
CHAR	is a keyword that declares the variable as type CHAR.
int2	is an integer that specifies the length in characters of a type CHAR variable.
DATE	is a keyword that declares the variable as type DATE.
DECIMAL	is a keyword that declares the variable as type DECIMAL.
int3,int4	are two integers that describe a type DECIMAL variable. These integers are not used by ACE and are accepted only to provide compatibility with RDSQL. You can omit them without any effect.

FLOAT	is a keyword that declares the variable as type FLOAT.
INTEGER	is a keyword that declares the variable as type INTEGER.
MONEY	is a keyword that declares the variable as type MONEY.
SMALLFLOAT	is a keyword that declares the variable as type SMALLFLOAT.
SMALLINT	is a keyword that declares the variable as type SMALLINT.

Notes

1. You can define a total of 100 variables using PARAM and VARIABLE statements in an ACE report specification.

2. If you want to use a variable that you declare in a PARAM or VARIABLE statement in the SELECT section, you must precede the variable name with a dollar sign. (Refer to the "SELECT Section" later in this chapter for more information.)

3. See "Database Data Types" in Chapter 2, "Using RDSQL," for a complete discussion of variable data types.

Examples

The following example is from the **ord3.ace** report:

```
define
   variable begin_date date
   variable end_date date
end
```

The values for the two variables are entered by the user when the program runs. The "INPUT Section" that follows describes this process.

INPUT Section

An ACE report specification optionally can contain an INPUT section. The INPUT section allows you to produce an interactive ACE report by prompting for and accepting input while ACE is running a report.

The INPUT section consists of the keywords INPUT and END with one or more PROMPT FOR statements in between. The structure of the INPUT section is shown below.

```
INPUT
   PROMPT FOR statement(s)
END
```

PROMPT FOR

Overview

This statement prompts you while ACE is running a report and assigns the value you enter to a variable.

Syntax

PROMPT FOR *var-name* USING "*string*"

Explanation

PROMPT FOR are required keywords.

var-name	is the name of the variable that receives your input. You must declare this variable in the DEFINE section of the ACE specification.
USING	is a required keyword.
string	is the string of characters that ACE uses as a prompt. You must enclose this string in quotation marks.

Notes

1. You cannot prompt for, or accept, a database name using the PROMPT FOR statement. Refer to the DATABASE section and to the discussion of the **-d** option in the "Command Line Options" section earlier in this chapter.

2. You cannot prompt for, or accept, an output filename using the PROMPT FOR statement.

Examples

The following example is from the **ord3.ace** report:

```
input
   prompt for begin_date
   using "Enter beginning date for report: "

   prompt for end_date
   using "Enter ending date for report: "
end
```

The two character strings, "Enter beginning date for report:" and "Enter ending date for report:," appear as prompts on the screen when the **ord3.ace** report runs. The response to the first prompt is entered as the value to the **begin_date** variable; the response to the second prompt is entered as the value to the **end_date** variable. These two variables are used at several points in the **ord3.ace** report.

OUTPUT Section

An ACE report specification can optionally contain an OUTPUT section. The OUTPUT section controls the width of the margins and the length of the page. The OUTPUT section also allows you to direct the output from the ACE report to a file or the system printer. On UNIX systems, you can direct output to a pipe.

The OUTPUT section begins with the OUTPUT keyword and ends with the corresponding END keyword, with one or more statements in between. The structure of the OUTPUT section is shown below.

```
OUTPUT
    REPORT TO statement
    LEFT MARGIN statement
    RIGHT MARGIN statement
    TOP MARGIN statement
    BOTTOM MARGIN statement
    PAGE LENGTH statement
END
```

REPORT TO

Overview

This statement directs the output of the ACE report to a file, the system printer, or (on UNIX systems only) a pipe.

Syntax

REPORT TO { "*filename*" | PRINTER | PIPE "*program*" }

Explanation

REPORT TO	are required keywords.
filename	is the name of a system file that receives the report. You must enclose the filename in quotation marks.
PRINTER	is the keyword that sends the report to the printer.
PIPE	is the keyword that sends the report to a pipe.
program	is the name of a system command.

Notes

1. When you do not use one of the REPORT TO statements, ACE sends the report to your screen.

2. You cannot use more than one REPORT TO statement in a report specification.

3. The TO PRINTER keywords cause ACE to send the report to the program named by the DBPRINT environment variable. If you do not define this environment variable, ACE sends the report to the **lp** program (on UNIX systems) or the **lpt1** printer (on DOS systems).

4. If you want to send the report to a printer other than the system printer, you can use the REPORT TO *filename* statement to send the output to a file and then send the file to the printer of your choice.

5. If the REPORT TO *filename* statement outputs to an existing filename, the file is replaced with the new output. On UNIX systems you can also use the REPORT TO PIPE statement to direct the output to a program that will send the output to the appropriate printer.

6. The TO PIPE keywords are effective only on UNIX systems.

Examples

The following example directs the output to the **labels** file:

```
output
   report to "labels"
end
```

The following example directs the output from an ACE report to the **more** utility:

```
output
   report to pipe "more"
end
```

LEFT MARGIN

Overview

This statement sets a left margin for a report.

Syntax

LEFT MARGIN *integer*

Explanation

LEFT MARGIN	are required keywords.
integer	is an integer that specifies the width of the left margin in spaces.

Notes

1. The default left margin is five spaces.

Examples

The following example is from the **mail2.ace** report. ACE prints the left side of the report as far to the left as possible.

```
output
    top margin 0
    bottom margin 0
    left margin 0
    page length 9
    report to "labels"
end
```

RIGHT MARGIN

Overview

This statement sets a right margin for a report.

Syntax

RIGHT MARGIN *integer*

Explanation

RIGHT MARGIN	are required keywords.
integer	is an integer that specifies the width of the text on the page in characters.

Notes

1. The RIGHT MARGIN determines the right margin by specifying the width of the page in characters. It is not dependent on the LEFT MARGIN, but always starts its count from the left edge of the page (space 0).

2. The RIGHT MARGIN is only effective when the FORMAT section contains an EVERY ROW statement.

3. The default right margin is 132 characters.

Examples

The following example report specification demonstrates the use of the RIGHT MARGIN statement. ACE sets the right margin for the report at 70 characters.

```
database
    stores
end

output
    right margin 70
end

select *
    from customer
end

format
    every row
end
```

TOP MARGIN

Overview

This statement sets a top margin for a report.

Syntax

TOP MARGIN *integer*

Explanation

TOP MARGIN are required keywords.

integer is an integer that specifies the number of blank lines that ACE leaves at the top of each page.

Notes

1. The default top margin is three lines.

2. The top margin appears above any page header you specify.

Examples

The following example is from the **mail2.ace** report. ACE begins printing at the top of each page.

```
output
    top margin 0
    bottom margin 0
    left margin 0
    page length 9
    report to "labels"
end
```

BOTTOM MARGIN

Overview

This statement sets a bottom margin for a report.

Syntax

BOTTOM MARGIN *integer*

Explanation

BOTTOM MARGIN	are required keywords.
integer	is an integer that specifies the number of blank lines that ACE is to leave at the bottom of each page.

Notes

1. The default bottom margin is three lines.
2. The bottom margin appears below any page trailer.

Examples

In the following example, the printing continues to the bottom of each page:

```
output
   top margin 0
   bottom margin 0
end
```

PAGE LENGTH

Overview

This statement sets the number of lines on each page of a report.

Syntax

PAGE LENGTH *integer*

Explanation

PAGE LENGTH	are required keywords.
integer	is an integer that specifies the length of the page in lines.

Notes

1. The default page length is sixty-six lines.

2. The PAGE LENGTH includes the TOP MARGIN and BOTTOM MARGIN.

Examples

The following example demonstrates the use of the PAGE LENGTH statement. ACE prints each page with 22 lines.

```
output
   {This length works on std 24—line crt}
   page length 22
   top margin 0
   bottom margin 0
end
```

SELECT Section

Every report specification must have a SELECT section. The SELECT section specifies the columns and/or tables the report is based on. In addition, you can use the SELECT section to specify criteria for selecting and ordering rows based on the contents of specific columns. The FORMAT section can group rows in the report based on the ordering you specify in the SELECT section.

The SELECT section contains one or more SELECT statements. These statements are identical to the RDSQL SELECT statements described in Chapter 7, "Using RDSQL," of the *INFORMIX-SQL User Guide*, and Chapter 2, "The RDSQL Query Language," of this manual. This chapter does not define the SELECT statement, but shows how to incorporate it in an ACE report specification.

The SELECT section begins with the SELECT keyword. This keyword introduces both the SELECT section *and* the first SELECT statement. Other SELECT statements can follow the first—each must begin with the SELECT keyword. All SELECT statements, except for the last, must end with a semicolon. (If there is only one SELECT statement, it does not require a semicolon.) The SELECT section ends with the END keyword. All but the last SELECT statement must have an INTO TEMP clause; only the last SELECT statement may have an ORDER BY clause.

If you use an ORDER BY clause in the SELECT section, you may not use an integer nor a column with a table prefix (**table.column**) to indicate the column to sort by. If you cannot use the column name alone because it is not unique or because it is an expression, define a display label in the *select-list* and use it in both the ORDER BY clause and the FORMAT section in the AFTER and BEFORE GROUP OF control blocks. The second example in this section demonstrates the use of a display label.

If you use an ACE variable in the SELECT section, you must precede the variable name with a dollar sign. The third example in this section demonstrates the use of a variable name.

The following shows the structure of a SELECT section.

```
SELECT statement;
  .
  .
  .
SELECT statement
END
```

Examples

The following example is from the **mail1.ace** report. ACE selects all rows from the **customer** table and orders the rows first by zip code and then by last name.

```
select *
  from customer
  order by zipcode, lname
end
```

The following example is from the **ord1.ace** report:

```
select
    orders.order_num number,
    order_date, customer_num,
    po_num, ship_date, ship_charge,
    paid_date,

    items.order_num, stock_num, manu_code,
    quantity, total_price

    from orders, items

    where orders.order_num = items.order_num

    order by number
end
```

ACE selects the indicated columns from the **orders** and **items** tables. The **order_num** column in the **orders** table is given the display label **number** and is joined to the **order_num** column in the **items** table. ACE orders the rows by the values in the **number** column.

The following example is from the **clist2.ace** report:

```
select
        customer_num,
        fname,
        lname,
        company,
        city,
        state,
        zipcode,
        phone
   from
        customer
   where
        state matches $thisstate
   order by
        zipcode,
        lname
end
```

ACE selects the indicated columns from the **customer** table. The WHERE clause tells ACE to select only those rows where the value in the **state** column matches the value in the variable **thisstate**. ACE orders the rows by the values in the **zipcode** column first and then by **lname**.

FORMAT Section

An ACE report specification must contain a FORMAT section. The FORMAT section determines how the report appears. It works with the data that is qualified by the last (or only) SELECT statement in the SELECT section. The FORMAT section begins with the FORMAT keyword and ends with the corresponding END keyword.

There are two major types of FORMAT sections. The simplest FORMAT section contains only an EVERY ROW statement between the FORMAT and END keywords. If you use an EVERY ROW statement, you cannot use any other statements or control blocks in the FORMAT section. The following shows the structure of this type of FORMAT section.

```
FORMAT
   EVERY ROW statement
END
```

More complex FORMAT sections can contain control blocks such as ON EVERY ROW and BEFORE GROUP OF. Each of these control blocks must contain at least one statement such as PRINT or SKIP *n* LINES.

In the FORMAT section, you may not refer to a column using its table name (table.column) as a prefix. If you do not use an EVERY ROW statement, you can combine control blocks as required. You can place control blocks in any order within the FORMAT section. The following shows the structure of this type of FORMAT section.

```
FORMAT
    PAGE HEADER control block
    PAGE TRAILER control block
    FIRST PAGE HEADER control block
    ON EVERY ROW control block
    ON LAST ROW control block
    BEFORE GROUP OF control block(s)
    AFTER GROUP OF control block(s)
END
```

EVERY ROW

Overview

The EVERY ROW statement causes ACE to output every row that the SELECT section qualifies. It uses a default format.

Syntax

```
EVERY ROW
```

Explanation

EVERY ROW are required keywords.

Notes

1. This statement is useful when you want to develop a report quickly using a default format. The report uses as column headings the column names you assigned when you created the table. Because the EVERY ROW statement cannot contain any control blocks or other statements, you cannot alter the default format to create a custom report.

2. The EVERY ROW statement stands by itself—you cannot modify it with any of the statements listed in the "Statements" section of this chapter.

3. When you use the EVERY ROW statement, you cannot use any control blocks in the FORMAT section.

4. A report generated by an EVERY ROW statement uses the column names you assigned when you created the table as column headings.

5. If the columns you specify in the SELECT section fit on a line, ACE will produce a report with column names across the top of each page; otherwise it will produce a report with the column names down the left side of the page.

6. You can use the RIGHT MARGIN statement in the OUTPUT section to control the width of a report that uses the EVERY ROW statement.

7. Use the ON EVERY ROW control block if you want to display every row in a format other than the default format. (See the discussion of ON EVERY ROW in the "Control Blocks" section later in this chapter.)

Examples

The following example shows a minimal ACE report specification using the EVERY ROW statement.

```
database
    stores
end

select *
    from customer
end

format
    every row
end
```

The following shows a portion of the output from the preceding specification.

```
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
      .
      .
      .
```

Below is another example of a brief report specification that uses the EVERY ROW statement.

```
database
    stores
end

select order_num, customer_num,
    order_date
    from orders
end

format
    every row
end
```

The following shows the output from the preceding specification.

```
order_num customer_num order_date

     1001          104  01/20/1986
     1002          101  06/01/1986
     1003          104  10/12/1986
     1004          106  04/12/1986
     1005          116  12/04/1986
     1006          112  09/19/1986
     1007          117  03/25/1986
     1008          110  11/17/1986
     1009          111  02/14/1986
     1010          115  05/29/1986
     1011          104  03/23/1986
     1012          117  06/05/1986
     1013          104  09/01/1986
     1014          106  05/01/1986
     1015          110  07/10/1986
```

Control Blocks

Control blocks provide the structure for a custom report. The control blocks that you can use in a FORMAT section are listed below:

- AFTER GROUP OF
- BEFORE GROUP OF
- FIRST PAGE HEADER
- ON EVERY ROW
- ON LAST ROW
- PAGE HEADER
- PAGE TRAILER

Each control block is optional, but if you do not use the EVERY ROW statement, you must include at least one control block in a report specification.

Each control block must include at least one statement (see the "Statements" section later in this chapter). If you have **INFORMIX-ESQL/C**, you can also call C functions from within a control block. See the *INFORMIX-ESQL/C Programmers Manual* for details.

When you use an ORDER BY clause in the SELECT section of an ACE report specification, you can use BEFORE GROUP OF and AFTER GROUP OF control blocks in the FORMAT section. When you use the BEFORE GROUP OF, AFTER GROUP OF, and ON EVERY ROW control blocks in a single report specification, ACE processes the control blocks in the order shown in Figure 5-2. (Assume that the SELECT section orders by columns **a**, **b**, and **c**.)

```
before group of a
        before group of b
                before group of c
                        on every row
                after group of c
        after group of b
after group of a
```

Figure 5-2. Order of Group Processing

AFTER GROUP OF

Overview

The AFTER GROUP OF control block specifies what action ACE is to take after it processes a group of rows. Grouping is determined by the ORDER BY clause of the SELECT section.

Syntax

AFTER GROUP OF
column-name[*int1*[,*int2*]]
statement(s)

Explanation

AFTER GROUP OF are required keywords.

column-name is the name of one of the columns specified in the ORDER BY clause of the SELECT section.

int1,int2 are optional subscripts that you can use with a type CHAR column to refer to a subset of the column beginning at character position *int1* and ending at character position *int2*. If *int2* is not specified, the end of the column is used. You must enclose subscripts in [ ] brackets.

statement(s) is a list of one or more statements or a compound statement.

Notes

1. A *group* of rows is all rows that contain the same value for a given column. ACE automatically groups rows when you use an ORDER BY clause in the SELECT section of a report specification—that is, groups come together when you order a list.

   When you specify more than one column in an ORDER BY clause, ACE orders the rows first by the first column you specify (most significant), second by the second column you specify, and so on, until the last column you specify (least significant).

   ACE processes the statements in an AFTER GROUP OF control block each time the specified column changes value, each time a more significant column changes value, and at the end of a report (see Figure 5-2 at the beginning of the "Control Blocks" section).

2. You can have one AFTER GROUP OF control block for each column specified in the ORDER BY clause in the SELECT section.

3. If you have more than one AFTER GROUP OF control block, their order within the FORMAT section is not significant. ACE will process the AFTER GROUP OF control blocks in the reverse order specified in the ORDER BY clause in the SELECT section (see Figure 5-2).

4. When ACE finishes generating a report, it executes all of the statements in the AFTER GROUP OF control blocks before it executes those in the ON LAST ROW control block.

5. You cannot reference the *column-name* in the AFTER GROUP OF clause using the name of a table (the *table.column* structure is not allowed). If you cannot use the column name alone because it is not unique or because it is an expression, define a display label in the *select-list* and use it in the AFTER GROUP OF clause.

6. You cannot use an integer to indicate by which column of the *select-list* the grouping is to occur.

7. Group aggregates can be used only in AFTER GROUP OF control blocks. You cannot use group aggregates in any other control blocks.

8. If you specify a substring of a CHAR column in an ORDER BY clause in the SELECT section, you must use the same substring specification as the *column-name* in an AFTER GROUP OF control block.

9. You can use a SKIP TO TOP OF PAGE statement in an AFTER GROUP OF control block to start a new page after each group.

10. When ACE processes the statements in an AFTER GROUP OF control block, the columns that the report is processing still have the values from the last row of the group. From this perspective, the AFTER GROUP OF control block could be called the "on last row of group" control block.

Examples

The following example is from the **ord2.ace** report:

```
after group of number

skip 1 line
print 4 spaces, "Shipping charges for the order:  ",
 ship_charge using "$$$$.&&"
skip 1 line

print 5 spaces, "Total amount for the order:  ",
   ship_charge + group total of total_price
   using "$$,$$$,$$$.&&"
skip 3 lines

after group of custnum
skip 2 lines
```

BEFORE GROUP OF

Overview

The BEFORE GROUP OF control block specifies what action ACE is to take before it processes a group of rows. Grouping is determined by the ORDER BY clause of the SELECT section.

Syntax

```
BEFORE GROUP OF
  column-name[int1[,int2]]
  statement(s)
```

Explanation

BEFORE GROUP OF	are required keywords.
column-name	is the name of one of the columns specified in the ORDER BY clause of the SELECT section.
int1,int2	are optional subscripts that you can use with a type CHAR column to refer to a subset of the column beginning at character position *int1* and ending at character position *int2*. If *int2* is not specified, the end of the column is used. You must enclose subscripts in [ ] brackets.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. A *group* of rows is all rows that contain the same value for a given column. ACE automatically groups rows when you use an ORDER BY clause in the SELECT section of a report specification—that is, groups come together when you order a list.

   When you specify more than one column in an ORDER BY clause, ACE orders the rows first, by the first column you specify (most significant), second, by the second column you specify, and so on, until the last column you specify (least significant).

   ACE processes the statements in a BEFORE GROUP OF control block at the start of a report, each time the specified column changes value, and each time a more significant column changes value (see Figure 5-2 at the beginning of the "Control Blocks" section).

2. You can have one BEFORE GROUP OF control block for each column specified in the ORDER BY clause in the SELECT section.

3. If you have more than one BEFORE GROUP OF control block, their order within the FORMAT section is not significant. ACE will process the BEFORE GROUP OF control blocks in the reverse order specified in the ORDER BY clause in the SELECT section (see Figure 5-2).

4. When ACE starts to generate a report, it executes all of the statements in the BEFORE GROUP OF control blocks before it executes those in the ON EVERY ROW control block.

5. You cannot reference the *column-name* in the BEFORE GROUP OF clause using the name of a table (the *table.column* structure is not allowed). If you cannot use the column name alone because it is not unique or because it is an expression, define a display label in the *select-list* and use it in the BEFORE GROUP OF clause.

6. You cannot use an integer to indicate by which column of the *select-list* the grouping is to occur.

7. If you specify a substring of a CHAR column in an ORDER BY clause in the SELECT section, you must use the same substring specification as the *column-name* in a BEFORE GROUP OF control block.

8. You can use a SKIP TO TOP OF PAGE statement in a BEFORE GROUP OF control block to start a new page after each group.

9. When ACE processes the statements in a BEFORE GROUP OF control block, the columns that the report is processing still have the values from the last row of the group. From this perspective, the BEFORE GROUP OF control block could be called the "on first row of group" control block.

Examples

The following example is from the **ord1.ace** report:

```
before group of ordnum

    print "Order number:  ", ordnum using "#####",
       " for customer number:  ", customer_num
       using "#####"
    print "Customer P.O. :  ", po_num,
       "  Date ordered:  ", order_date

    skip 1 line
    print "Stockno", column 20,
       "Mfcode", column 28, "Qty", column 38, "Price"
```

FIRST PAGE HEADER

Overview

The FIRST PAGE HEADER control block specifies what information appears at the top of the first page of the report.

Syntax

FIRST PAGE HEADER *statement(s)*

Explanation

FIRST PAGE HEADER	are required keywords.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. The vertical size of the first page header is equal to the number of lines that you specify in the FIRST PAGE HEADER control block. The TOP MARGIN (in the OUTPUT section) affects how close to the top of the page ACE displays the page header.

2. A FIRST PAGE HEADER control block overrides a PAGE HEADER control block on the first page of a report.

3. You cannot use the SKIP TO TOP OF PAGE statement in a FIRST PAGE HEADER control block.

4. If you use an IF THEN ELSE statement in a FIRST PAGE HEADER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.

5. You cannot use the PRINT FILE statement to read and display text from a file in a FIRST PAGE HEADER control block.

6. You can use a FIRST PAGE HEADER control block to produce a title page as well as column headings.

Examples

The following example is from the **mail3.ace** report:

```
first page header

let i = 1

let l_size = 72/count1

let white = 8/count1
```

This FIRST PAGE HEADER does not display any information. Because ACE executes the FIRST PAGE HEADER control block *before* it generates any output, you can use this control block (as demonstrated in the example) to initialize variables that you use in the FORMAT section.

ON EVERY ROW

Overview

The ON EVERY ROW control block specifies what action ACE is to take after the SELECT section qualifies a row.

Syntax

ON EVERY ROW *statement(s)*

Explanation

ON EVERY ROW	are required keywords.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. ACE processes the statements in an ON EVERY ROW control block as each new row is formatted.

2. If a BEFORE GROUP OF control block is triggered by a change in column value, all BEFORE GROUP OF control blocks will be executed (in the order of their significance) before the ON EVERY ROW control block is executed.

3. If an AFTER GROUP OF control block is triggered by a change in column value, all AFTER GROUP OF control blocks will be executed (in the order of their significance) after the ON EVERY ROW control block is executed.

Examples

The following example is from the **clist1.ace** report:

```
on every row
     print customer_num using "####",
     column 9, fname clipped, 1 space, lname clipped
     column 32, city clipped, ", " , state,
     column 54, zipcode,
     column 62, phone
```

The following example is from the **mail2.ace** report:

```
on every row
    if (city is not null) and
       (state is not null) then
    begin
       print fname clipped, 1 space, lname
       print company
       print address1
       if (address2 is not null) then
          print address2
       print city clipped, ", " , state,
       2 spaces, zipcode
       skip to top of page
    end
```

ON LAST ROW

Overview

The ON LAST ROW control block specifies the action ACE takes after processing the last row qualified by the SELECT section.

Syntax

ON LAST ROW *statement(s)*

Explanation

ON LAST ROW	are required keywords.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. ACE executes the statements in the ON LAST ROW control block after it executes those in the ON EVERY ROW and AFTER GROUP OF control blocks.

2. You can use the ON LAST ROW control block to display report totals.

3. When ACE processes the statements in an ON LAST ROW control block, the values from the last row returned by the SELECT statement are current and can be used.

Examples

The following example is from the **clist1.ace** report:

```
on last row
     skip 1 line
     print  "TOTAL NUMBER OF CUSTOMERS:",
          column 30, count using "##"
```

PAGE HEADER

Overview

The PAGE HEADER control block specifies what information will appear at the top of each page of the report.

Syntax

PAGE HEADER *statement(s)*

Explanation

PAGE HEADER	are required keywords.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. The vertical size of the page header is equal to the number of lines that you specify in the PAGE HEADER control block. The TOP MARGIN (in the OUTPUT section) affects how close to the top of the page ACE displays the page header.

2. A FIRST PAGE HEADER control block overrides a PAGE HEADER control block on the first page of a report.

3. You cannot use the SKIP TO TOP OF PAGE statement in a PAGE HEADER control block.

4. If you use an IF THEN ELSE statement in a PAGE HEADER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.

5. If you use a FOR or WHILE statement that contains a PRINT statement in a PAGE HEADER control block, you must terminate the print statement with a semicolon. The semicolon will suppress any NEWLINEs (RETURNs) in the loop, keeping the number of lines in the header constant from page to page.

6. You cannot use a PRINT FILE statement to read and display text from a file in a PAGE HEADER control block.

7. You can use a PAGE HEADER control block to display column headings in a report.

8. You can use the PAGENO expression in a PRINT statement within a PAGE HEADER control block to display the page number automatically at the top of every page.

Examples

The following example is from the **clist1.ace** report:

```
page header
     print "NUMBER",
       column 9, "NAME",
       column 32, "LOCATION",
       column 54, "ZIP",
       column 62, "PHONE"
     skip 1 line
```

PAGE TRAILER

Overview

The PAGE TRAILER control block specifies what information will appear at the bottom of each page of the report.

Syntax

PAGE TRAILER *statement(s)*

Explanation

PAGE TRAILER	are required keywords.
statement(s)	is a list of one or more statements or a compound statement.

Notes

1. The vertical size of the page trailer is equal to the number of lines that you specify in the PAGE TRAILER control block. The BOTTOM MARGIN (in the OUTPUT section) affects how close to the bottom of the page ACE displays the page trailer.

2. You cannot use the SKIP TO TOP OF PAGE statement in a PAGE TRAILER control block.

3. If you use an IF THEN ELSE statement in a PAGE TRAILER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.

4. If you use a FOR or WHILE statement that contains a PRINT statement in a PAGE TRAILER control block, you must terminate the PRINT statement with a semicolon. The semicolon will suppress NEWLINES (RETURNS) in the loop, keeping the number of lines in the trailer constant from page to page.

5. You cannot use the PRINT FILE statement to read and display text from a file in a PAGE TRAILER control block.

6. You can use the PAGENO expression in a PRINT statement within a PAGE TRAILER control block to display the page number automatically at the bottom of every page.

Examples

The following example is from the **ord3.ace** report:

```
page trailer
   print column 28, pageno using "page <<<<"
```

Statements

The format control blocks determine *when* ACE takes an action, while the statements determine *what* action ACE takes.

Statements are composed of keywords and expressions, as explained under each of the specific statements. A subsequent section of this chapter covers expressions.

Any statement can be a single statement or a compound statement. A compound statement is one or more statements, including other compound statements, preceded by the BEGIN keyword and followed by an END keyword.

FOR

Overview

The FOR statement defines a loop. It repeatedly executes a simple or compound statement, incrementing the loop index before each pass through the loop. Control passes to the first statement following the end of the loop when the termination condition is satisfied.

Syntax

FOR *loop-index* = *expr1* TO *expr2* [STEP *expr3*]
  DO *statement*

Explanation

FOR	is a required keyword.
loop-index	is the name of a variable declared in the DEFINE section. The FOR statement uses this variable as the loop index, changing its value each time through the loop.
=	is a required keyword.
expr1	is a required expression that specifies the starting value of the loop index.
TO	is a required keyword.
expr2	is a required expression that specifies the ending value of the loop index. This expression specifies the termination condition for the loop.
STEP	is an optional keyword.

expr3	is an optional expression that specifies the amount the FOR statement will increment the loop index each time through the loop. If you do not specify STEP and *expr3*, the FOR statement will assume an increment value of 1.
DO	is a required keyword.
statement	is a single statement or a compound statement.

Notes

1. You cannot have a decrementing loop—the value of *expr3* must be positive.

2. If a compound statement follows the DO keyword, you must precede the compound statement with a BEGIN keyword and follow it with END.

IF THEN ELSE

Overview

This statement defines a conditional branch. It evaluates an expression and executes specific statements based on the result of the evaluation.

Syntax

IF *expr* THEN *statement1*
[ELSE *statement2*]

Explanation

IF	is a required keyword.
expr	is a required expression that determines which, if any, of the statements IF executes.
THEN	is a required keyword.
statement1	is a required single statement or compound statement that IF executes if *expr* evaluates as true (not equal to zero).
ELSE	is an optional keyword.
statement2	is an optional single statement or compound statement that IF executes if *expr* evaluates as false (equal to zero).

Notes

1. If a compound statement follows the THEN or ELSE keyword, you must precede the compound statement with the BEGIN keyword and follow it with END.

2. You can nest IF THEN ELSE statements to 128 levels.

Examples

The following example is from the **mail3.ace** report:

```
if i = count1 then
   begin

      print array1 clipped
      print array2 clipped
      print array3 clipped

      skip 1 line

      let array1 = " "
      let array2 = " "
      let array3 = " "

      let i  = 1
   end

else
   let i = i + 1
```

LET

Overview

The LET statement assigns a value to a declared variable.

Syntax

LET *variable*[*num-expr*[,*num-expr*]] = *expr-list*

Explanation

LET	is a required keyword.
variable	is a required variable name; that is, a variable declared in a previous DEFINE section.
num-expr	is an optional numeric expression or list of one or two numeric expressions. They specify a substring of the CHAR variable to which LET is to assign a value. If one *num-expr* is specified, the substring is the beginning character within the variable through the end of the string. If two *num-expr* are present, the substring is from the first *num-expr* through the second one. You can only use these substring operations with a CHAR variable. You must enclose subscripts in [ ] brackets.
=	is a required keyword.
expr-list	is a required list of one or more expressions separated by commas. LET assigns the value of this list to the *variable*.

If the list contains more than one expression, the *variable* must be of type CHAR. LET will assign the value of the string generated by the concatenation of all expressions in the expression list to the variable. The result that ACE assigns to the variable takes the same form as if you had displayed the same expression list with a PRINT statement. LET accepts all of the expressions that PRINT accepts including USING, CLIPPED, ASCII, COLUMN, and subscripted character expressions.

Notes

1. If you assign a value with a fractional part to an INTEGER or SMALLINT variable, ACE truncates the fractional part.

2. Refer to the "Expressions" section of this chapter for more information about type conversion.

Examples

The following example is from the FORMAT Section of the **mail3.ace** report:

```
let i = 1

let l_size = 72/count1

let white = 8/count1
```

NEED

Overview

This statement causes subsequent display to start on the next page if the specified number of lines cannot be placed on the current page.

Syntax

NEED *num-expr* LINES

Explanation

NEED	is a required keyword.
num-expr	is an expression that evaluates to a number specifying the number of lines needed.
LINES	is a required keyword.

Notes

1. Use the NEED statement to prevent ACE from separating parts of the report that you want to keep together on a single page.

PAUSE

Overview

This statement causes output to the terminal to pause until you press **RETURN**.

Syntax

PAUSE [*string*]

Explanation

PAUSE	is a required keyword.
string	is an optional message that PAUSE displays. If you do not supply a message, PAUSE will not display a message.

Notes

1. The PAUSE statement will have no effect if you use a REPORT TO *filename*, a REPORT TO PRINTER, or a REPORT TO PIPE (on UNIX systems only) statement in the OUTPUT section.

Examples

The following example causes ACE to pause while running the report.

```
after group of item_num
     .
     .
     .
skip to top of page
pause "Press RETURN to continue"
```

PRINT

Overview

This statement displays information on the screen or as specified in the OUTPUT section.

Syntax

PRINT [*expr-list*][;]

Explanation

PRINT	is a required keyword.
expr-list	is an optional list of one or more expressions, separated by commas.
;	is an optional keyword that suppresses a NEWLINE (RETURN) at the end of the line.

Notes

1. One PRINT statement displays its output on one line, no matter how many lines the statement occupies in the report specification.

2. Unless you use the keyword CLIPPED or USING following an expression, ACE displays an expression so that it occupies a predetermined number of spaces. (See Figure 5-3.)

Column	Default Size
CHAR	declared size
DATE	10
FLOAT	14 (including sign and decimal point)
SMALLINT	6 (including sign)
INTEGER	11 (including sign)
SMALLFLOAT	14 (including sign and decimal point)
DECIMAL	number of digits plus 2 (including sign and decimal point)
SERIAL	11
MONEY	number of digits plus 3 (including sign, decimal point, and dollar sign)

Figure 5-3. Default Display Widths

Examples

The following example is from the **clist2.ace** report:

```
first page header
print column 32, "CUSTOMER LIST"
print column 32, "_____"
skip 2 lines
print "Listings for the State of ", thisstate
skip 2 lines
print "NUMBER",
     column 9, "NAME",
     column 32, "LOCATION",
     column 54, "ZIP",
     column 62, "PHONE"
skip 1 line

page header
print "NUMBER",
     column 9, "NAME",
     column 32, "LOCATION",
     column 54, "ZIP",
     column 62, "PHONE"
skip 1 line

on every row
print customer_num using "####",
      column 9, fname clipped, 1 space, lname clipped
      column 32, city clipped, ", " , state,
      column 54, zipcode,
      column 62, phone
```

PRINT FILE

Overview

This statement displays the contents of a text file in a report.

Syntax

PRINT FILE *"filename"*

Explanation

PRINT FILE	are required keywords.
filename	is a required filename that can be a pathname. You must enclose the filename in quotation marks.

Notes

1. You can use the PRINT FILE statement to include the body of a form letter in a report that generates custom letters.

SKIP

Overview

This statement skips lines in a report.

Syntax

SKIP *int* LINES

Explanation

SKIP	is a required keyword.
int	is an integer number specifying the number of lines to skip.
LINES	is a required keyword. You can use the keyword LINE in place of LINES if you like.

Examples

The following example is from the **mail1.ace** report:

```
format
  on every row
     print fname, lname
     print company
     print address1
     print address2
     print city, ", " , state,
           2 spaces, zipcode
     skip 2 lines
```

SKIP TO TOP OF PAGE

Overview

This statement causes subsequent printing to begin at the top of the next page.

Syntax

```
SKIP TO TOP OF PAGE
```

Explanation

SKIP TO TOP OF PAGE are required keywords.

Notes

1. You cannot use a SKIP TO TOP OF PAGE statement in a FIRST PAGE HEADER, PAGE HEADER, or PAGE TRAILER control block.

Examples

The following example is from the **mail2.ace** report:

```
format
on every row
   if (city is not null) and
      (state is not null) then
   begin
      print fname clipped, 1 space, lname
      print company
      print address1
      if (address2 is not null) then
         print address2
      print city clipped, ", " , state,
         2 spaces, zipcode
      skip to top of page
    end
end
```

WHILE

Overview

The WHILE statement defines a loop which repeatedly executes a simple or compound statement while the expression is true. Control passes to the first statement following the loop when the expression evaluates as false.

Syntax

WHILE *expr* DO
statement

Explanation

WHILE	is a required keyword.
expr	is a required expression. While this expression evaluates as true, WHILE executes the loop. When this expression evaluates as false, control passes to the first statement following the loop.
DO	is a required keyword.
statement	is a single statement or a compound statement

Notes

1. If more than one statement follows the DO keyword, you must precede them with the BEGIN keyword and follow them with END to create a compound statement.

Aggregates

Overview

Aggregates allow you to summarize information in a report.

Syntax

```
[GROUP]
   {COUNT | PERCENT |
   {TOTAL | {AVERAGE | AVG} | MIN | MAX} OF expr1}}
 [WHERE expr2]
```

Explanation

GROUP	is an optional keyword that causes the aggregate to reflect information for a specific group only. The group must have been designated by an ORDER BY clause in the SELECT section of the report specification. You can only use this keyword in an AFTER GROUP OF control block.
COUNT	is a keyword that is evaluated as the total number of rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.
PERCENT	is the keyword that evaluates COUNT as a percent of the total number of rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.

TOTAL	evaluates as the total of *expr1* in the rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.
AVERAGE/AVG	evaluates as the average of *expr1* in the rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.
MIN	evaluates as the minimum of *expr1* in the rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.
MAX	evaluates as the maximum of *expr1* in the rows qualified by the SELECT section and the optional WHERE *expr2* expression that can appear in an aggregate statement.
OF	is a keyword.
expr1	is the expression that TOTAL, AVERAGE, MIN, and MAX evaluate. It is typically a column or a numeric expression that includes a numeric column.
WHERE	is a keyword.
expr2	is a logical expression that qualifies the aggregate.

Notes

1. The WHERE part of an aggregate statement further qualifies rows that the SELECT section already qualified. It cannot select rows that were not qualified by the SELECT section.

2. Aggregates produce unpredictable results when *expr1* or *expr2* contains user-defined variables (see "PARAM" and "VARIABLE" in this chapter).

Examples

The following example is from the **ord2.ace** report:

```
on every row
    print snum using "###", column 10, manu_code,
       column 18, description clipped, column 38,
       quantity using "###", column 43, unit_price
       using "$$$$.&&", column 55,
       total_price using "$$,$$$,$$$.&&"

after group of number

    skip 1 line

    print 4 spaces,
       "Shipping charges for the order:  ",
       ship_charge using "$$$$.&&"

    skip 1 line

    print 5 spaces, "Total amount for the order:  ",
       ship_charge + group total of
       total_price using "$$,$$$,$$$.&&"
```

ASCII

Overview

ACE evaluates this expression as a value that you can use as a character. You can use it to display CONTROL characters.

Syntax

ASCII *num-expr*

Explanation

ASCII	is a required keyword.
num-expr	is a numeric expression.

Examples

The following PRINT statement rings the terminal bell (ASCII value of 7).

```
print ascii 7
```

The next report specification segments show how to implement special printer or terminal functions. They assume that when the printer receives the sequence of ASCII characters 9, 11, and 1, it will start printing in red, and when it receives 9, 11, and 0, it will revert to black printing. The values used in the example are hypothetical; refer to your printer or terminal manual for information on your printer or terminal.

This specification uses the FIRST PAGE HEADER control block to initialize variables that are used in other control blocks.

```
    .
    .
define
      variable red_on  char(3)
      variable red_off char(3)
end
    .
    .
format
   first page header
      let red_on  =
          ascii 9, ascii 11, ascii 1
      let red_off =
          ascii 9, ascii 11, ascii 0

   on every row
      .
      .
      print red_on,
          "Your bill is overdue.",
      red_off
        .
        .
```

Caution! ACE cannot distinguish printable and non-printable ASCII characters. Be sure to account for the non-printing characters when using the COLUMN expression to format your page. Since various devices differ in outputting spaces with control characters, you may have to use trial and error to line up columns when you output control characters.

CLIPPED

Overview

This expression displays the character field that precedes it without any trailing blanks.

Syntax

---

char-expr CLIPPED

Explanation

char-expr	is a required character expression.
CLIPPED	is a required keyword.

Notes

1. You will normally use CLIPPED following a *column-name* in a PRINT statement.

Examples

The following example is from the **mail2.ace** report:

```
format
   on every row
      if (city is not null) and
         (state is not null) then
      begin
         print fname clipped, 1 space, lname
         print company
         print address1
         if (address2 is not null) then
            print address2
         print city clipped, ", " , state,
            2 spaces, zipcode
         skip to top of page
      end
end
```

COLUMN

Overview

This expression evaluates to a string of spaces long enough to position the next item in the designated column.

Syntax

COLUMN *num-expr*

Explanation

COLUMN	is a required keyword.
num-expr	is a required numeric expression that specifies the column position for the next item to be printed.

Notes

1. ACE calculates the column number by adding the number to the left margin you set in the OUTPUT section.

2. If ACE has already printed past the column specified by *num-expr*, ACE ignores the COLUMN expression.

Examples

The following example is from the **clist2.ace** report:

```
page header
print "NUMBER",
     column 9, "NAME",
     column 32, "LOCATION",
     column 54, "ZIP",
     column 62, "PHONE"
skip 1 line
```

DATE

Overview

This expression evaluates as a character string with a value of today's date in the form "Thu Sep 20 1984."

Syntax

DATE

Explanation

DATE is a required keyword.

Examples

Because DATE evaluates as type CHAR, you can use it with subscripts to express a day, month, date, or year. This exampl

```
print "Today is ", date[1,3]
```

displays the three-letter abbreviation for the day of the week

```
Today is Mon
```

DATE()

Overview

The DATE function converts the expression with which you call it to type DATE.

Syntax

DATE(*date-expr*)

Explanation

DATE	is a required keyword.
date-expr	is a required expression of any type that evaluates to a type DATE value.

Notes

1. The DATE function is typically used to convert date strings to type DATE.

DAY()

Overview

The DAY function returns the day of the month when you call it with a type DATE expression.

Syntax

DAY(date-expr)

Explanation

DAY	is a required keyword.
date-expr	is a required expression that evaluates to a type DATE value.

LINENO

Overview

This expression has the value of the line number of the line that ACE is currently displaying. ACE computes the line number by calculating the number of lines from the top of the page.

Syntax

```
LINENO
```

Explanation

LINENO is a required keyword.

Notes

Do not use LINENO within a page header or trailer. LINENO will work on the first page header, but will not work on any of the subsequent pages.

MDY()

Overview

The MDY function returns a type DATE value when you call it with three expressions that evaluate to integers representing the month, date, and year.

Syntax

MDY(*num-expr1*, *num-expr2*, *num-expr3*)

Explanation

MDY	is a required keyword.
num-expr1	is an expression that evaluates to an integer representing the number of the month (1–12).
num-expr2	is an expression that evaluates to an integer representing the number of the day of the month (1–28, 29, 30, or 31, depending on the month).
num-expr3	is an expression that evaluates to a four-digit integer representing the year.

MONTH()

Overview

The MONTH function returns an integer corresponding to the month (1-12) of its type DATE argument.

Syntax

MONTH(*date-expr*)

Explanation

MONTH	is a required keyword.
date-expr	is a required expression that evaluates to a type DATE value.

PAGENO

Overview

This expression has the value of the page number of the page that ACE is currently displaying.

Syntax

```
PAGENO
```

Explanation

PAGENO is a required keyword.

Notes

1. Use PAGENO in a PRINT statement in the PAGE HEADER or PAGE TRAILER control block to number the pages of a report. (You can also use PAGENO in other control blocks.)

Examples

The following example is from the **ord3.ace** report:

```
page trailer
   print column 28, pageno using "page <<<<"
```

SPACES

Overview

This expression evaluates as a string of spaces. It is identical to a quoted string of spaces.

Syntax

---

num-expr SPACE[S]

Explanation

num-expr	is a numeric expression.
SPACE[S]	is a required keyword. You can use either the keyword SPACE or SPACES.

Examples

The following example is from the **mail1.ace** report:

```
format
  on every row
     print fname, lname
     print company
     print address1
     print address2
     print city, ", " , state,
           2 spaces, zipcode
     skip 2 lines
  end
```

TIME

Overview

This expression evaluates as a character string with a value of the current time in the form *hh:mm:ss*.

Syntax

TIME

Explanation

TIME is a required keyword.

TODAY

Overview

This expression evaluates as type DATE with a value of the current date as supplied by the operating system.

Syntax

```
TODAY
```

Explanation

TODAY is a required keyword.

Examples

The following example is from the **ord3.ace** report:

```
skip 1 line
  print column 15, "FROM:  ", begin_date
     using "mm/dd/yy",
  column 35, "TO:  ", end_date
     using "mm/dd/yy"
  print column 15, "Report run date:  ",
     today using "mmm dd, yyyy"
  skip 2 lines
  print column 2, "ORDER DATE", column 15,
     "COMPANY", column 35, "NAME",
     column 57, "NUMBER", column 65, "AMOUNT"
```

USING

Overview

This expression allows you to format a numeric or date expression. With a numeric expression, you can use USING to line up decimal points, right- or left-justify numbers, put negative numbers in parentheses, and perform other formatting functions. With a date expression, USING converts the date to a variety of formats.

Syntax

---

expr1 USING *expr2*

Explanation

expr1	is the required expression that specifies what USING is to format.
USING	is a required keyword.
expr2	is the required format string that specifies how USING is to format *expr1*. Refer to "Notes" later in this section.

Formatting Numeric Expressions. The format string consists of combinations of the following characters: * & # < , . - + ( ) $. The characters - + ( ) $ will *float*. When a character floats, ACE displays multiple leading occurrences of the character as a single character as far to the right as possible, without interfering with the number that is being displayed. Refer to the following list.

* This character fills any positions with asterisks in the display field that would otherwise be blank.

& This character fills positions in with zeros in the display field that would otherwise be blank.

This character does not change any blank positions in the display field. You can use this character to specify a maximum width for a field.

< This character causes the numbers in the display field to be left-justified.

, This character is a literal; USING displays it as a comma. USING will not display a comma unless there is a number to its left.

. This character is a literal; USING displays it as a period. You can only have one period in a format string.

- This character is a literal; USING displays it as a minus sign when *expr1* is less than zero. When you group several in a row, a single minus sign floats to the rightmost position without interfering with the number being printed.

+ This character is a literal; USING displays it as a plus sign when *expr1* is greater than or equal to zero and as a minus sign when it is less than zero. When you group several in a row, a single plus sign floats to the rightmost position without interfering with the number being printed.

(This character is a literal; USING displays it as a left parenthesis before a negative number. It is the accounting parenthesis that is used in place of a minus sign to indicate a negative number. When you group several in a row, a single left parenthesis floats to the rightmost position without interfering with the number being printed.

) This is the accounting parenthesis that is used in place of a minus sign to indicate a negative number. A single one of these characters generally closes a format string that begins with a left parenthesis.

$ This character is a literal; USING displays it as a dollar sign. When you group several in a row, a single dollar sign

floats to the rightmost position without interfering with the number being printed.

Refer to the "Examples" section for examples of formatting numeric expressions.

Formatting Date Expressions. The format string consists of combinations of the characters *m*, *d*, and *y* shown in Figure 5-4.

dd	Day of the month as a 2-digit number (01–31)
ddd	Day of the week as a 3-letter abbreviation (Sun through Sat)
mm	Month as a 2-digit number (01–12)
mmm	Month as a 3-letter abbreviation (Jan through Dec)
yy	Year as a 2-digit number in the 1900s (00–99)
yyyy	Year as a 4-digit number (0001–9999)

Figure 5-4. Combinations of Date Format Strings

Refer to the "Examples" section for examples of formatting date expressions.

Notes

1. The format string must appear between quotation marks.

2. Although USING is generally used as part of a PRINT statement, you can also use it with LET.

3. If you attempt to display a number that is too large for a display field, ACE will fill the field with asterisks to indicate an overflow.

Examples

The example below prints the balance field using a format string that allows up to $9,999,999.99 to be formatted correctly.

```
print "The current balance is ",
   23485.23 using "$#,###,##&.&&"
```

The result of executing this PRINT statement with the value 23,485.23 is shown below.

```
The current balance is $  23,485.23
```

This example fixes the dollar sign. If dollar signs had been used instead of # characters, the dollar sign would have floated with the size of the number. It also uses a mix of # and & fill characters. The # character provides blank fill for unused character positions, while the & character provides zero filling. This format ensures that even if the number is zero, the positions marked with &s will appear as zeros, not blanks.

Example Format Strings

Format String	Numeric Value	Formatted Result
"#####"	0	bbbbb
"&&&&&"	0	00000
"$$$$$"	0	bbbb$
"*****"	0	*****
"<<<<<"	0	(null string)
"##,###"	12345	12,345
"##,###"	1234	b1,234
"##,###"	123	bbb123
"##,###"	12	bbbb12
"##,###"	1	bbbbb1
"##,###"	−1	bbbbb1
"##,###"	0	bbbbbb
"&&,&&&"	12345	12,345
"&&,&&&"	1234	01,234
"&&,&&&"	123	000123
"&&,&&&"	12	000012
"&&,&&&"	1	000001
"&&,&&&"	0	000000
"$$,$$$"	12345	****** (overflow)
"$$,$$$"	1234	$1,234
"$$,$$$"	123	bb$123
"$$,$$$"	12	bbb$12
"$$,$$$"	1	bbbb$1
"$$,$$$"	0	bbbbb$
"**,***"	12345	12,345
"**,***"	1234	*1,234
"**,***"	123	***123
"**,***"	12	****12
"**,***"	1	*****1
"**,***"	0	******

This table uses the character b to represent a blank or space.

Example Format Strings

Format String	Numeric Value	Formatted Result
"##,###.##"	12345.67	12,345.67
"##,###.##"	1234.56	b1,234.56
"##,###.##"	123.45	bbb123.45
"##,###.##"	12.34	bbbb12.34
"##,###.##"	1.23	bbbbb1.23
"##,###.##"	0.12	bbbbb0.12
"##,###.##"	0.01	bbbbb0.01
"##,###.##"	−0.01	bbbbb0.01
"&&,&&&.&&"	12345.67	12,345.67
"&&,&&&.&&"	1234.56	01,234.56
"&&,&&&.&&"	123.45	000123.45
"&&,&&&.&&"	0.01	000000.01
"$$,$$$.$$"	12345.67	********* (overflow)
"$$,$$$.$$"	1234.56	$1,234.56
"$$,$$$.##"	0.00	$.00
"$$,$$$.##"	1234.00	$1,234.00
"$$,$$$.&&"	0.00	$.00
"$$,$$$.&&"	1234.00	$1,234.00
"−##,###.##"	−12345.67	−12,345.67
"−##,###.##"	−123.45	−123.45
"−##,###.##"	−12.34	−12.34
"—#,###.##"	−12.34	−12.34
"——,###.##"	−12.34	−12.34
"——,−##.##"	−12.34	−12.34
"——,—#.##"	−1.00	−1.00
"−##,###.##"	12345.67	12,345.67
"−##,###.##"	1234.56	1,234.56
"−##,###.##"	123.45	123.45
"−##,###.##"	12.34	12.34
"—#,###.##"	12.34	12.34
"——,###.##"	12.34	12.34
"——,−##.##"	12.34	12.34
"——,——.##"	1.00	1.00
"——,——.—"	−.01	−.01

This table uses the character b to represent a blank or space.

Example Format Strings

Format String	Numeric Value	Formatted Result
"——,——.&&"	−.01	−.01
"−$$$,$$$.&&"	−12345.67	−$12,345.67
"−$$$,$$$.&&"	−1234.56	−$1,234.56
"−$$$,$$$.&&"	−123.45	−$123.45
"——$$,$$$.&&"	−12345.67	−$12,345.67
"——$$,$$$.&&"	−1234.56	−$1,234.56
"——$$,$$$.&&"	−123.45	−$123.45
"——$$,$$$.&&"	−12.34	−$12.34
"——$$,$$$.&&"	−1.23	−$1.23
"———,——$.&&"	−12345.67	−$12,345.67
"———,——$.&&"	−1234.56	−$1,234.56
"———,——$.&&"	−123.45	−$123.45
"———,——$.&&"	−12.34	−$12.34
"———,——$.&&"	−1.23	−$1.23
"———,——$.&&"	−.12	−$.12
"$***,***.&&"	12345.67	$*12,345.67
"$***,***.&&"	1234.56	$**1,234.56
"$***,***.&&"	123.45	$****123.45
"$***,***.&&"	12.34	$*****12.34
"$***,***.&&"	1.23	$******1.23
"$***,***.&&"	.12	$*******.12
"($$$,$$$.&&)"	−12345.67	($12,345.67)
"($$$,$$$.&&)"	−1234.56	($1,234.56)
"($$$,$$$.&&)"	−123.45	($123.45)
"(($$,$$$.&&)"	−12345.67	($12,345.67)
"(($$,$$$.&&)"	−1234.56	($1,234.56)
"(($$,$$$.&&)"	−123.45	($123.45)
"(($$,$$$.&&)"	−12.34	($12.34)
"(($$,$$$.&&)"	−1.23	($1.23)
"((((,(($.&&)"	−12345.67	($12,345.67)
"((((,(($.&&)"	−1234.56	($1,234.56)
"((((,(($.&&)"	−123.45	($123.45)
"((((,(($.&&)"	−12.34	($12.34)
"((((,(($.&&)"	−1.23	($1.23)
"((((,(($.&&)"	−.12	($.12)

Example Format Strings

Format String	Numeric Value	Formatted Result
"($$$,$$$.&&)"	12345.67	$12,345.67
"($$$,$$$.&&)"	1234.56	$1,234.56
"($$$,$$$.&&)"	123.45	$123.45
"(($$,$$$.&&)"	12345.67	$12,345.67
"(($$,$$$.&&)"	1234.56	$1,234.56
"(($$,$$$.&&)"	123.45	$123.45
"(($$,$$$.&&)"	12.34	$12.34
"(($$,$$$.&&)"	1.23	$1.23
"((((,(($.&&)"	12345.67	$12,345.67
"((((,(($.&&)"	1234.56	$1,234.56
"((((,(($.&&)"	123.45	$123.45
"((((,(($.&&)"	12.34	$12.34
"((((,(($.&&)"	1.23	$1.23
"((((,(($.&&)"	.12	$.12
"<<<,<<<"	12345	12,345
"<<<,<<<"	1234	1,234
"<<<,<<<"	123	123
"<<<,<<<"	12	12

The following example shows some sample conversions for December 25, 1985.

Format String	Formatted Result
"mmddyy"	122585
"ddmmyy"	251285
"yymmdd"	851225
"yy/mm/dd"	85/12/25
"yy mm dd"	85 12 25
"yy–mm–dd"	85–12–25
"mmm. dd, yyyy"	Dec. 25, 1985
"mmm dd yyyy"	Dec 25 1985
"yyyy dd mm"	1985 25 12
"mmm dd yyyy"	Dec 25 1985
"ddd, mmm. dd, yyyy"	Tue, Dec. 25, 1985
"(ddd) mmm. dd, yyyy"	(Tue) Dec. 25, 1985

WEEKDAY( )

Overview

The WEEKDAY function returns an integer that represents the day of the week when you call it with a type DATE expression.

Syntax

WEEKDAY(*date-expr*)

Explanation

WEEKDAY	is a required keyword.
date-expr	is a required expression that evaluates to a type DATE value.

Notes

1. WEEKDAY returns an integer in the range 0–6. 0 represents Sunday, 1 represents Monday, and so forth.

YEAR( )

Overview

The YEAR function returns an integer that represents the year when you call it with a type DATE expression.

Syntax

YEAR(*date-expr*)

Explanation

YEAR	is a required keyword.
date-exp	is a required expression that evaluates to a type DATE value.

Chapter 6

User-Menu

Chapter 6 Table of Contents

Overview 5

Accessing a Menu 6
Using a Menu Within INFORMIX-SQL 6

Designing a Menu 8

Creating a Menu 10
Accessing PERFORM with the **menuform** Form 10
Entering Menu Data 12
Steps for Entering Your Own Data 16

Modifying a Menu 18

Menu Display Fields 18
MENU NAME 19
MENU TITLE 20
SELECTION NUMBER 21
SELECTION TYPE 23
SELECTION TEXT 25
SELECTION ACTION 26

Creating a Script Menu 29

Overview

User-menu allows you to create and run custom menus. The options on a user-menu can call

- Submenus
- INFORMIX-SQL programs (PERFORM, for example)
- Other programs or sets of programs in your software library
- Operating system utilities
- Forms or reports

You use a special PERFORM screen form to create or alter a menu structure. Two special tables hold the menus, text, and command references for each menu option in the menu structure.

One user-menu can be created for each database. You cannot create a user-menu without specifying a database. The options in a user-menu, however, do not have to refer to any particular database.

A user-menu cannot exist separately from a database. If you want to keep the user-menu apart from your working databases, create a database that contains only menu data.

Caution! User-menu programs themselves require space in memory. Therefore, if you run large programs on DOS systems from the **User-menu**, it is possible to run out of memory. If your programs are too large to run from the **User-menu**, you must run them from either the appropriate INFORMIX-SQL menu option, or from the operating system command line.

If you are unable to run a program from the User-menu because of its size, exit to the command line. This action will clear memory. You can then resume your operations from either the operating system command line, or from the INFORMIX-SQL menu (by reentering isql from the command line).

This chapter contains six sections in addition to the Overview:

- Accessing a Menu
- Designing a Menu
- Creating a Menu
- Modifying a Menu
- Menu Display Fields
- Creating a Script Menu

The first section explains how to access a user menu. The second section discusses the design of a menu structure. The third section explains entering data into the menu form. The fourth section discusses how to change an existing user menu. The fifth section explains the kind of information to enter in each field in the **menuform** screen. The final section explains how to use the user-menu to perform a series of actions in sequence for a single menu item.

Accessing a Menu

A menu can be accessed from the INFORMIX-SQL Main Menu or from the operating system command line. The next section describes the use of the **User-menu** option on the INFORMIX-SQL Main Menu. Appendix M explains how to access a user-menu directly from the operating system command line.

Using a Menu Within INFORMIX-SQL

Follow these steps to access the user-menu included with the **stores** demonstration database.

1. Use the **Database** option on the INFORMIX-SQL Main Menu to make the **stores** database your current database.

2. Select the **User-menu** option on the INFORMIX-SQL Main Menu. The USER-MENU Menu is displayed.

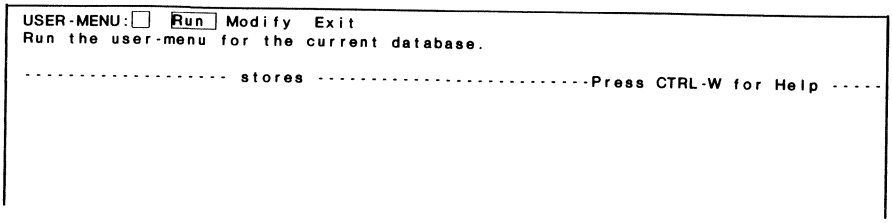


Figure 6-1. The USER-MENU Menu

3. Select the **Run** option on the USER-MENU Menu. The Main Menu of the user-menu for the **stores** database is displayed.

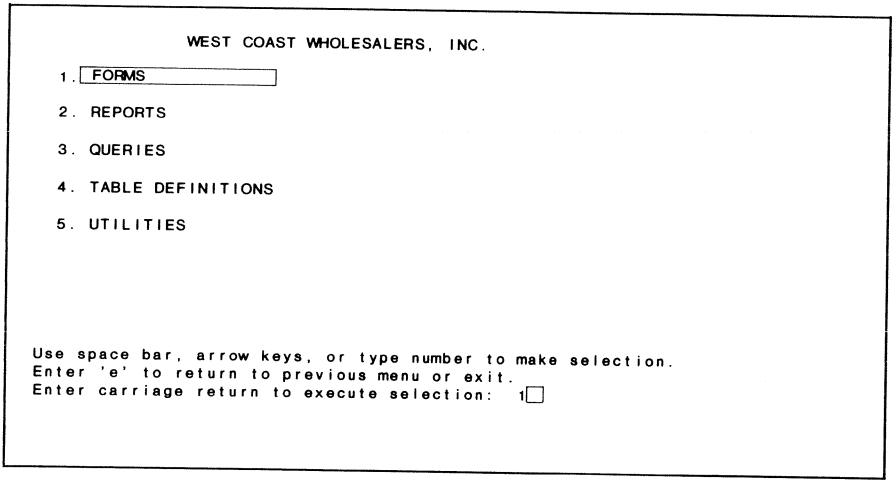


Figure 6-2. The Main Menu of the **stores** User-Menu

4. Select a menu option by typing the number to the left of the desired option or positioning the highlight on the option with the ARROW keys. Press the RETURN key. INFORMIX-SQL executes the option you select.

5. Type e to exit a current menu. Type e from the user-menu Main Menu, and the INFORMIX-SQL USER-MENU Menu is displayed.

Designing a Menu

You can have up to 19 levels of menus in the user-menu structure and up to 28 options on each menu.

The total number of options in a single menu depends on two factors:

- The number of lines your screen can hold
- The length of the menu titles

Most screens can accommodate 14 single-spaced menu lines. Each menu line can display two options of up to 33 characters of text in each option. If the text for each option on a menu does not exceed 33 characters , you can display up to 14 double-spaced menu options (two options per line) or 28 single-spaced menu options (two options per line). If the text for an option is longer than 33 characters, it requires an entire line, reducing the total number of options available for that menu.

As an example of a simple menu, consider the user-menu for the **stores** demonstration database. It has five options on the top or main level:

- FORMS
- REPORTS
- QUERIES
- TABLE DEFINITIONS
- UTILITIES

Each option on the Main Menu calls a submenu, and each option on a submenu performs an action.

A design outline for the user-menu included with the **stores** database looks like this:

WEST COAST WHOLESALERS, INC.

1. USE DATA ENTRY FORMS
    1. CUSTOMER ENTRY/QUERY FORM
    2. ORDER ENTRY/QUERY FORM
    3. DISPLAY CUSTOMER FORM SPECIFICATION
    4. DISPLAY ORDER FORM SPECIFICATION

2. RUN REPORTS
    1. RUN CUSTOMER REPORT
    2. RUN REPORT ON CUSTOMER BY DESIGNATED STATE
    3. RUN CUSTOMER MAILING LABELS
    4. RUN REPORT ON ORDERS BETWEEN SELECTED DATES
    5. RUN PROMOTIONAL LETTERS
    6. RUN MATRIX REPORT ON MONTHLY SALES
    7. DISPLAY REPORT 1 SPECIFICATION
    8. DISPLAY REPORT 2 SPECIFICATION
    9. DISPLAY REPORT 3 SPECIFICATION
    10. DISPLAY REPORT 4 SPECIFICATION
    11. DISPLAY REPORT 5 SPECIFICATION
    12. DISPLAY REPORT 6 SPECIFICATION

3. EXECUTE DATABASE QUERIES
    1. ENTER NEW RDSQL QUERIES
    2. DISPLAY CUSTOMER INFORMATION SORTED BY CITY AND LASTNAME
    3. DISPLAY INFORMATION ON PRICE INCREASE
    4. DISPLAY CUSTOMER INFORMATION BASED ON PARTIAL NAME MATCH
    5. INSERT, UPDATE, SELECT, AND DELETE NEW CUSTOMER ROW
    6. DISPLAY ALL CURRENTLY UNPAID ORDERS
    7. JOIN THREE TABLES TO DISPLAY INFORMATION ON STOCK
    8. DISPLAY CREATION OF VIEW WITH PRIVILEGES
    9. DISPLAY INFORMATION BASED ON A VIEW
    10. DISPLAY DROPPING OF VIEW
    11. DISPLAY CUSTOMERS PLUS OUTSTANDING ORDERS (NO OUTER JOIN)
    12. DISPLAY CUSTOMERS PLUS OUTSTANDING ORDERS (OUTER JOIN)

4. DISPLAY DATABASE TABLE DEFINITIONS
    1. TABLE "CUSTOMER"
    2. TABLE "ITEMS"
    3. TABLE "MANUF"
    4. TABLE "ORDERS"
    5. TABLE "STOCK"

5. UTILITY MENU
    1. DISPLAY DATE AND TIME
    2. CHECK CUSTOMER TABLE
    3. UNLOAD CUSTOMER TABLE

Creating a Menu

The first step in creating a menu is to access the **PERFORM** program with the **menuform** screen form. The second step is to enter data through this screen form.

The **menuform** form is a special **PERFORM** screen form that you use only to create or to modify a user-menu. You can enter, change, and remove menu information with the form, but you cannot change the appearance of the screen form itself, and you cannot run **FORMBUILD** on it.

Accessing PERFORM with the **menuform** *Form*

Select the **User-menu** option on the INFORMIX-SQL Main Menu. If there is no current database, the CHOOSE DATABASE Screen appears. After you select a database, the USER-MENU Menu is displayed. Select the **Modify** option and **INFORMIX-SQL** displays the PERFORM menu with the **menuform** form.

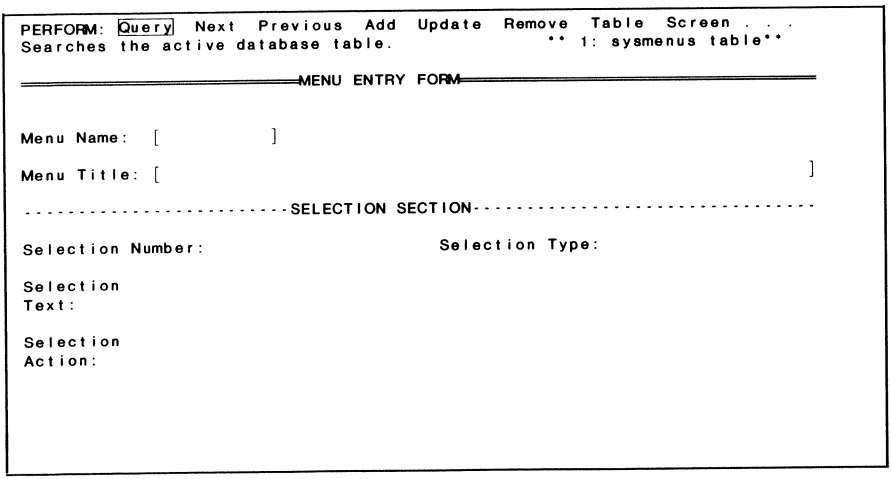


Figure 6-3. The PERFORM Menu with the **menuform** Form

Two tables are accessed in the **menuform** form:

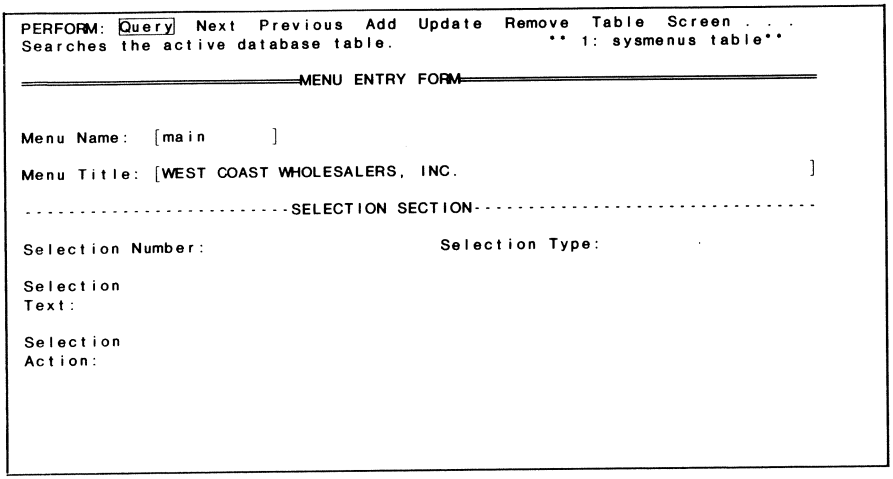
- The **sysmenus** table stores information about each menu in the user-menu. This includes the Menu Name (used by INFORMIX-SQL to identify the menu), and the Menu Title (text that you want to appear when the menu is displayed).

- The **sysmenuitems** table stores information about the options on each menu. This includes the option number, the option type (program, report, form, RDSQL command file, script menu, or submenu), the option title (text that appears on the screen), and the action the option specifies (execute a program, report, form or script menu, or call an RDSQL command file or a submenu). A script menu allows the user to run multiple actions in sequence for a single item. Script menus are described in the section "Creating a Script Menu" later in this chapter.

The **sysmenus** table is the master of the **sysmenuitems** table.

Entering Menu Data

You enter menu data in two steps. First you enter data for a menu in the two fields at the top of the screen form (the fields associated with the **sysmenus** table). For example, the entry into the **sysmenus** table for the Main Menu in the **stores** database is shown in this screen:



The information stored in the **sysmenus** table in the **stores** database appears in the following display:

Menu Name	**Menu Title**
main	WEST COAST WHOLESALERS, INC.
forms	FORMS
reports	REPORTS
queries	DATABASE QUERIES
tables	DATABASE TABLE DEFINITIONS
utilities	UTILITIES

Second, you enter data for each option of each menu in the fields on the lower half of the screen (the fields associated with the **sysmenuitems** table).

You must make an entry in the **sysmenus** table before you make entries for a menu option in the **sysmenuitems** table. This is necessary because INFORMIX-SQL checks when you enter data in the **sysmenuitems** table to make sure that menus cited in that table exist in the **sysmenus** table.

For example, the complete entry for option 1 on the Main Menu is shown in this screen:

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

                          MENU ENTRY FORM

Menu Name:  [main                ]

Menu Title:  WEST COAST WHOLESALERS, INC.

------------------------SELECTION SECTION-------------------------------

Selection Number:  [1       ]          Selection Type:  [M]

Selection
Text:       [FORMS                                                      ]

Selection
Action:     [forms                                                      ]
```

In this instance, the Selection Action (forms) is the name of a menu (corresponding to Selection Type M). Information about the **forms** menu must have been entered into the **sysmenus** table prior to completing the entry for option 1.

The information stored in the **sysmenuitems** table in the **stores** database appears in the following table:

Menu Name	Option	Selection Text	Type	Action
main	1	FORMS	M	forms
main	2	REPORTS	M	reports
main	3	QUERIES	M	queries
main	4	TABLE DEFINITIONS	M	tables
main	5	UTILITIES	M	utilities
forms	1	CUSTOMER ENTRY/QUERY FORM	F	customer
forms	2	ORDER ENTRY/QUERY FORM	F	orders
forms	3	DISPLAY CUSTOMER FORM SPECIFICATION	P	cat customer.per
forms	4	DISPLAY ORDER FORM SPECIFICATION	P	cat orders.per
reports	1	RUN CUSTOMER REPORT	R	clist1
reports	2	RUN REPORT ON CUSTOMER BY DESIGNATED STATE	R	clist2
reports	3	RUN CUSTOMER MAILING LABELS	S	mailinglabels
reports	4	RUN REPORT ON ORDERS BETWEEN SELECTED DATES	S	dailyorders
reports	5	RUN PROMOTIONAL LETTERS	S	letters
reports	6	RUN MATRIX REPORT ON MONTHLY SALES	R	months
reports	7	DISPLAY REPORT1 SPECIFICATION	P	cat clist1.ace
reports	8	DISPLAY REPORT2 SPECIFICATION	P	cat clist2.ace
reports	9	DISPLAY REPORT3 SPECIFICATION	P	cat mail.ace
reports	10	DISPLAY REPORT4 SPECIFICATION	P	cat dailyord.ace
reports	11	DISPLAY REPORT5 SPECIFICATION	P	cat letters.ace
reports	12	DISPLAY REPORT6 SPECIFICATION	P	cat months.ace
mailinglabels	1	run mailing labels report	R	mail
mailinglabels	2	display output file from mailing labels report	P	cat mail.out
dailyorders	1	run dailyord report	R	dailyord
dailyorders	2	display output file from dailyorders report	P	cat dailyord.out
letters	1	run letters report	R	letters
letters	2	display output file from letters report	P	cat letters.out
queries	1	ENTER NEW RDSQL QUERIES	Q	
queries	2	DISPLAY CUSTOMER INFORMATION SORTED BY CITY AND LASTNAME	S	query2
queries	3	DISPLAY INFORMATION ON PRICE INCREASE	S	query3
queries	4	DISPLAY CUSTOMER INFORMATION BASED ON PARTIAL NAME MATCH	S	query4
queries	5	INSERT, UPDATE, SELECT, AND DELETE NEW CUSTOMER ROW	S	query5
queries	6	DISPLAY ALL CURRENTLY UNPAID ORDERS	S	query6

Data in the **sysmenuitems** Table *(continued)*

queries	7	JOIN THREE TABLES TO DISPLAY INFORMATION ON STOCK	S	query7
queries	8	DISPLAY CREATION OF VIEW WITH PRIVILEGES	P	cat view__c.sql
queries	9	DISPLAY INFORMATION BASED ON A VIEW	S	query9
queries	10	DISPLAY DROPPING OF VIEW	P	cat view__d.sql
queries	11	DISPLAY CUSTOMERS PLUS OUTSTANDING (NO OUTER JOIN)	S	query11
queries	12	DISPLAY CUSTOMERS PLUS OUTSTANDING ORDERS (OUTER JOIN)	S	query12
query2	1	display RDSQL syntax for query menu choice 2	P	cat cust__cty.sql
query2	2	run query menu choice 2	Q	cust__cty
query3	1	display RDSQL syntax for query menu choice 3	P	cat p__inc.sql
query3	2	run query menu choice 3	Q	p__inc
query4	1	display RDSQL syntax for query menu choice 4	P	cat cust__nme.sql
query4	2	run query menu choice 4	Q	cust__nme
query5	1	display RDSQL syntax for query menu choice 5	P	cat cust__row.sql
query5	2	run query menu choice 5	Q	cust__row
query6	1	display RDSQL syntax for query menu choice 6	P	cat unpaid.sql
query6	2	run query menu choice 6	Q	unpaid
query7	1	display RDSQL syntax for query menu choice 7	P	cat stock.sql
query7	2	run query menu choice 7	Q	stock
query9	1	display RDSQL syntax for query menu choice 9	P	cat view__s.sql
query9	2	run query menu choice 9	Q	view__s
query11	1	display RDSQL syntax for query menu choice 11	P	cat orders1.sql
query11	2	run query menu choice 11	Q	orders1
query12	1	display RDSQL syntax for query menu choice 12	P	cat orders2.sql
query12	2	run query menu choice 12	Q	orders2
tables	1	TABLE "CUSTOMER"	P	cat c__custom.sql
tables	2	TABLE "ITEMS"	P	cat c__items.sql
tables	3	TABLE "MANUF"	P	cat c__manuf.sql
tables	4	TABLE "ORDERS"	P	cat c__orders.sql
tables	5	TABLE "STOCK"	P	cat c__stock.sql
utilities	1	DISPLAY DATE AND TIME	P	date
utilities	2	CHECK CUSTOMER TABLE	S	utility2
utilities	3	UNLOAD CUSTOMER TABLE	S	utility3
utility2	1	display RDSQL syntax for utility menu choice 2	P	cat ch__cust.sql
utility2	2	run utility menu choice 2	Q	ch__cust
utility3	1	display RDSQL syntax for utility menu choice 3	P	cat u__cust.sql
utility3	2	run utility menu choice 3	Q	u__cust
utility3	3	display output of utility menu coice 3	P	cat customer.unl

Steps for Entering Your Own Data

The following steps describe the procedure for entering your own menu and option data. See the list of display fields in the section "Menu Display Fields" later in this chapter for information on the kind of data to enter for each display field.

1. Select the **User-menu** option on the INFORMIX-SQL Main Menu.

2. Select the **Modify** option on the USER-MENU Menu. The **menuform** screen form is displayed.

3. Type a to select the **Add** option.

4. Enter a Menu Name and Menu Title for the first menu. Note that main must be the first Menu Name. Press ESC when you finish.

5. Enter data in the same way for the second menu. Press ESC when you finish.

6. When you have entered the Menu Name and Menu Title data for all menus in the User Menu, you are ready to enter data into the SELECTION SECTION of the form. You should now enter information about all options on the Main Menu. Select the **Query** option , enter main in the Menu Name field, and press ESC.

7. Type d to make the detail table (**sysmenuitems**) active. INFORMIX-SQL displays

   ```
   There are no rows satisfying the conditions
   ```

   The **sysmenuitems** table contains no rows joining the main entry and the Menu Name field.

8. Select the **Add** option. Enter the Selection Number, Selection Type, Selection Text, and Selection Action data for the first option on the Main Menu. Press ESC when you finish entering data about the first menu option. If there is a second option to the Main Menu, select **Add** and enter data about that option. Press ESC when you finish.

   Repeat this step until you have entered data for each option on the Main Menu.

9. Type m to call the Master table again. Use the **Query** option to locate and display the Menu Name and Menu Title data for your next menu. Type d to display the detail table joined to the current row of the Master table.

10. Enter the Selection Number, Selection Type, Selection Text, and Selection Action data for the first option on this menu. Press ESC when you finish entering data. Repeat this step for each option in this menu.

11. Repeat Steps 9 and 10 to enter data for the remaining menu options. When you have entered data for all the options in each menu, the menu is complete. Select the **Exit** option to leave PERFORM and return to the USER-MENU Menu.

12. Select the **Run** option on the USER-MENU Menu to run the new menu.

Modifying a Menu

You change a user-created menu in the same way you create one. Select the **User-menu** option from the INFORMIX-SQL Main Menu. Then select the **Modify** option on the USER-MENU Menu. Use the **PERFORM** options to modify the menu entries in the **menuform** screen form.

See Chapter 4, "The PERFORM Screen Transaction Processor," for information about **PERFORM**.

Menu Display Fields

This section discusses the kinds of information you can enter for each field in the **menuform** screen.

MENU NAME

Overview

INFORMIX-SQL uses the entry in the Menu Name field to find the menu you want when you make a selection that calls another menu. The menu name is used only by INFORMIX-SQL and is never displayed on a screen.

Notes

1. The menu name must follow the standard rules for identifiers. It can be from 1 to 18 characters long; the first character must be a letter; and you can use numbers, letters, and underscores (_) for the rest of the name.

2. The top-level menu must be named main in all lowercase letters.

Examples

The Menu Name entry for the Main Menu must be main:

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

                              MENU ENTRY FORM

Menu Name:  [main       ]

Menu Title: [                                                          ]

-------------------------SELECTION SECTION-----------------------------

Selection Number:                          Selection Type:

Selection
Text:

Selection
Action:
```

MENU TITLE

Overview

Use this field to enter the text INFORMIX-SQL displays at the top of the menu.

Notes

1. The brackets on the screen show the maximum length of the text. It can contain any number of words that fit within the brackets.

Examples

The Menu Title entry for the Main Menu follows:

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

                              MENU ENTRY FORM

Menu Name:  [main       ]

Menu Title: [WEST COAST WHOLESALERS, INC.                                  ]

------------------------SELECTION SECTION-------------------------------

Selection Number:                        Selection Type:

Selection
Text:

Selection
Action:
```

SELECTION NUMBER

Overview

Use the Selection Number field to enter the option number you want to appear to the left of each menu item on the screen.

Notes

1. INFORMIX-SQL displays the menu items in numbered order. The user selects items by number.

2. The total number of options you can have in one menu depends on two factors: the number of lines your screen can hold and the length of the menu titles you enter. Most screens can accommodate 14 single-spaced menu lines, and each menu line can display two options of up to 33 characters . If the text for each option on a menu does not exceed 33 characters , you can display up to 14 double-spaced menu options (two options per line) or 28 single-spaced menu options (two options per line). If the text for an option is longer than 33 characters, it requires an entire line, reducing the total number of options available for that menu.

Examples

The Selection Number entry for an option on the REPORTS Menu follows:

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

=========================MENU ENTRY FORM=========================

Menu Name:  [reports            ]

Menu Title:  REPORTS

------------------------SELECTION SECTION-----------------------------

Selection Number:  [3      ]            Selection Type:  [S]

Selection
Text:      [RUN CUSTOMER MAILING LABELS                              ]

Selection
Action:    [mailinglabels                                            ]



Enter the selection number for this item
```

SELECTION TYPE

Overview

Use the Selection Type field to specify the type of action an option performs. You can indicate that an option runs a form or report; calls a submenu; executes an RDSQL command file, a program, or an operating system command; or invokes a script menu.

Explanation

The options available for the Selection Type field are as follows:

F runs a form.

R runs a report.

M calls a submenu.

Q executes an RDSQL command file.

P executes a program or an operating system command.

S executes a script menu.

Notes

1. The entry in the Selection Type field must agree with the entry in the Selection Action field. For example, when the Selection Type is **R,** the Selection Action must be the name of a compiled report.

2. You can enter the Selection Type option in either an uppercase or lowercase letter. INFORMIX-SQL automatically displays it as an uppercase letter on the screen.

Examples

The Selection Type entry for running the **clist2** report follows:

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

=========================MENU ENTRY FORM=========================

Menu Name:  [reports              ]

Menu Title:  REPORTS

------------------------SELECTION SECTION-----------------------------

Selection Number:  [2       ]          Selection Type:  [R]

Selection
Text:      [RUN REPORT ON CUSTOMER BY DESIGNATED STATE                  ]

Selection
Action:    [clist2                                                     ]



Choices:  (M)enu  (P)rogram  (Q)uery-language  (R)eport  (F)orm  (S)cript
```

The Selection Type entry for running the **customer** entry form follows:

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

=========================MENU ENTRY FORM=========================

Menu Name:  [forms                ]

Menu Title:  FORMS

------------------------SELECTION SECTION-----------------------------

Selection Number:  [1       ]          Selection Type:  [F]

Selection
Text:      [CUSTOMER ENTRY/QUERY FORM                                  ]

Selection
Action:    [customer                                                   ]



Choices:  (M)enu  (P)rogram  (Q)uery-language  (R)eport  (F)orm  (S)cript
```

SELECTION TEXT

Overview

Use the Selection Text field to enter the text you want to appear to the right of the option number on the screen.

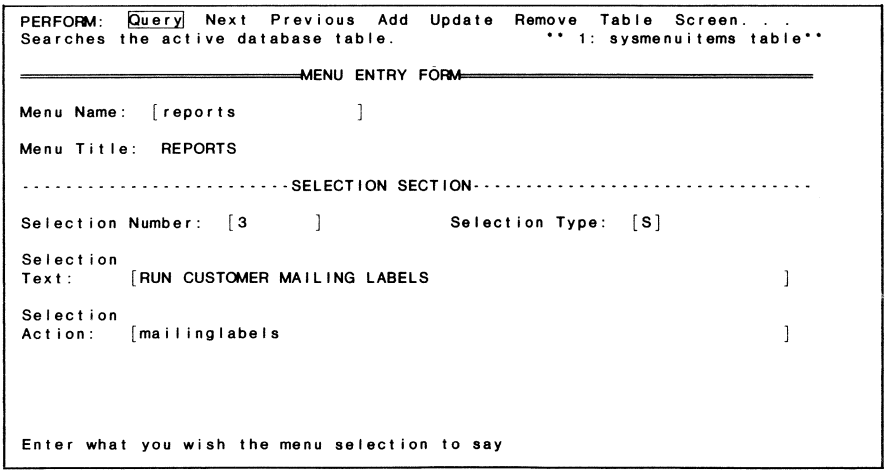
Notes

1. The brackets on the screen show the maximum length of the text allowed in this field.

2. The length of the selection text affects the total number of options you can include in a single menu. See the "Notes" section of "SELECTION NUMBER" for more information.

Examples

The Selection Text entry for an option on the REPORTS Menu follows:



SELECTION ACTION

Overview

Use the Selection Action field to specifiy the name of the action executed when the user selects the option indicated in the Selection Type field. You can enter a compiled form or report specification, an RDSQL command file, a submenu, an operating system command, a program, or a script menu.

Notes

1. The entry in the Selection Action field must agree with the entry in the Selection Type field:

Selection Type	Selection Action
M	Enter the menu name (not the menu title) of the submenu. You cannot enter a submenu name that does not exist in the Menu Name field in the **sysmenus** table. **Example:** reports
P	Enter a program name or an operating system command. **Example:** date
F	Enter a form name. It is not necessary to add the **.frm** extension. **Example:** customer
R	Enter a report name. It is not necessary to add the **.arc** extension. **Example:** clist1

Q	Enter an **RDSQL** command filename. It is not necessary to add the **.sql** extension. **Example:** cust_city
S	Enter a script menu name. **Example:** mailinglabels

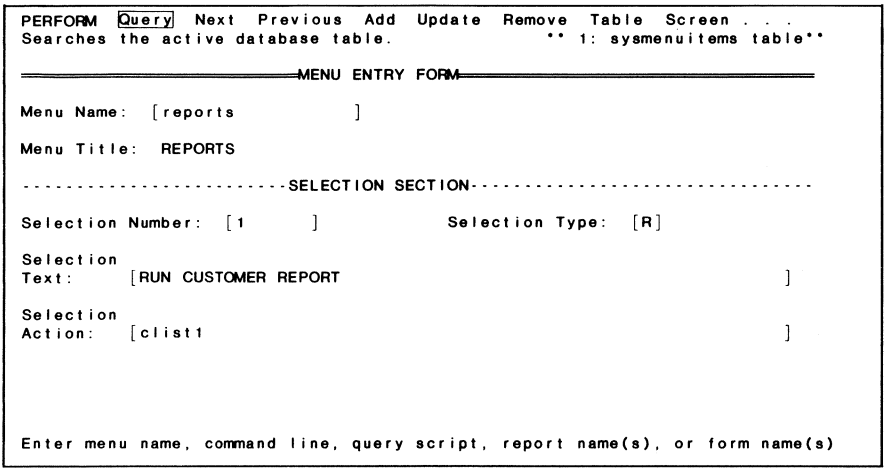
2. You can enter Q in the Selection Type field and nothing in the Selection Action field. When the user selects that option, **INFORMIX-SQL** calls the Query-language option on the **INFORMIX-SQL** Main Menu. The user can then enter one or more **RDSQL** statements.

3. You can enter an R or F in the Selection Type field and enter nothing in the Selection Action field. When the user selects this option, **INFORMIX-SQL** calls the **Report** or **Form** options, respectively, on the **INFORMIX-SQL** Main Menu.

4. When you finish using the **Query-language, Report,** or **Form** option, choose the **E** option to exit. **INFORMIX-SQL** then returns you to the User-Menu.

Examples

The Selection Action entry used in the demonstration database for running the **clist1** report follows:



The Selection Type R specifies run a report. The Selection Action specifies **clist1** as the name of the report to be run.

Creating a Script Menu

A script menu is a menu that includes more than one action for a single menu item. By selecting the appropriate menu option, these actions are run in sequence without the necessity of displaying the menu on the screen. An **S** in the Selection Type field requires the entry of a script menu name in the Selection Action field.

The following list describes the procedure used to create the *mailinglabels* script included in the demonstration user-menu. This script runs and displays customer mailing labels. This script is Selection Number 3 on the **Reports** submenu. When the user selects option 3 on the **Reports** submenu, the mailing labels report runs, and the output file is displayed on the screen.

1. Select the **Modify** option on the USER-MENU Menu. (See the section "Creating a Menu" earlier in this chapter.)

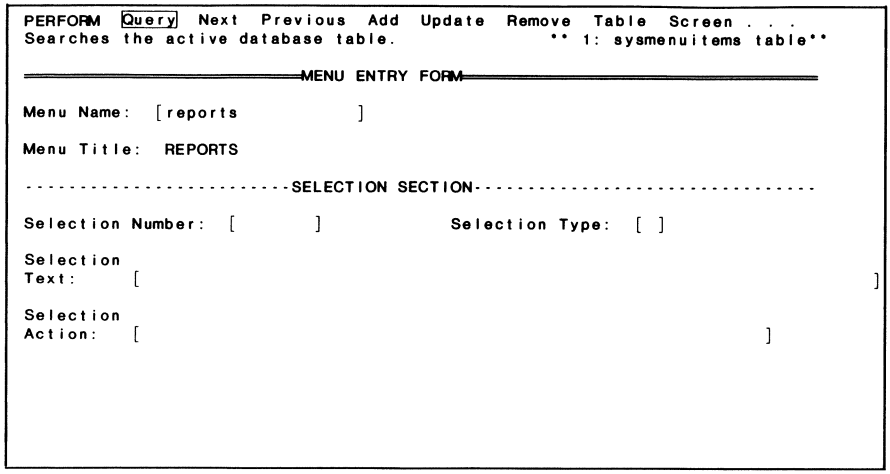
2. Type **a** to select the **Add** option.

3. Enter reports in the Menu Name field. The script becomes an option on the **Reports** submenu.

4. Enter REPORTS in the Menu Title field.

5. Press ESC when you finish.

The preceding steps show you how to enter the necessary information in the **sysmenus** table. The following display shows how the PERFORM screen appears at this point:



The following steps show you how to enter data into the lower half of the screen (the **sysmenuitems** table fields):

1. Type **d** to make the **detail** table active.

2. Type **a** to **Add.**

3. Enter 3 in the Selection Number field. The script becomes the third choice on the **Reports** submenu.

4. Enter **S** in the Selection Type field. This indicates you will run a script menu.

5. Enter

   RUN CUSTOMER MAILING LABELS

   in the Selection Text field. This is the text that appears to the right of the option number on the screen.

6. Enter mailinglabels in the Selection Action field. This is the name of the script menu you want to run.

7. Press ESC when you finish.

The following screen is the **PERFORM** screen as it appears at this point:

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

                          MENU ENTRY FORM

Menu Name:  [reports          ]

Menu Title:  REPORTS

------------------------SELECTION SECTION-----------------------------

Selection Number:  [3      ]          Selection Type:  [S]

Selection
Text:      [RUN CUSTOMER MAILING LABELS                                ]

Selection
Action:    [mailinglabels                                              ]
```

You must now enter the actions you want the script to perform and the order in which you want them to be performed.

1. Type **m** for **Master** to make the **sysmenus** table active.

2. Type **a** to select the **Add** option.

3. Enter mailinglabels in the Menu Name field. This is the name of the script menu.

4. Enter

   run report menu selection 3 and display the output file

   in the Menu Title field.

   Unlike all other user-menu menus, the entry in the Menu Title field for a script menu is not displayed on the screen. You can use it as a Comment Line to list the series of actions that comprise the script menu.

5. Press **ESC** when you finish.

The preceding steps show you how to enter the necessary information in the **sysmenus** table. The following display shows how the **PERFORM** screen appears at this point:

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

                          MENU ENTRY FORM

Menu Name:  [mailinglabels      ]

Menu Title:  run report menu selection 3 and display the output file

-----------------------SELECTION SECTION-------------------------------

Selection Number:  [        ]          Selection Type:  [ ]

Selection
Text:      [                                                          ]

Selection
Action:    [                                                          ]
```

The following steps show you how to enter data for each option on the menu into the lower half of the screen (the **sysmenuitems** table fields):

1. Type **d** to make the **detail** table active.

2. Select the **Add** option.

3. Enter 1 in the Selection Number field. **INFORMIX-SQL** executes this action first.

4. Enter R in the Selection Type field. This runs a report.

5. Enter run mailing labels report in the Selection Text field.

   Unlike other user-menu menus, the entry in the Selection Text field for a script menu is not displayed on the screen. You can use it as a Comment Line to describe the action specified by the entry in the Selection Type field.

6. Enter mail in the Selection Action field. This is the name of the compiled report specified by the action.

7. Press **ESC** when you finish.

The following screen is the **PERFORM** screen as it appears at this point:

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

                          MENU ENTRY FORM

Menu Name:  [mailinglabels      ]

Menu Title:  run report menu selection 3 and display the output file

------------------------SELECTION SECTION-------------------------------

Selection Number:  [1       ]          Selection Type:  [R]

Selection
Text:      [run mailing labels report                                   ]

Selection
Action:    [mail                                                        ]
```

You enter the second action for the *mailinglabels* script in a similar fashion to what you did when setting up the initial action.

1. Type **a** for Add.

2. Enter **2** in the Selection Number field. INFORMIX-SQL executes this action second.

3. Enter P in the Selection Type field. This executes a program.

4. Enter

   ```
   display output file from mailing labels report
   ```

   in the Selection Text field.

5. Enter cat mail.out in the Selection Action field. This is the name of the operating system program you want to run.

6. Press ESC when you finish.

The following screen is the completed **PERFORM** screen as it appears at this point:

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

═══════════════════════MENU ENTRY FORM═══════════════════════

Menu Name:  [mailinglabels      ]

Menu Title:  run report menu selection 3 and display the output file

------------------------SELECTION SECTION-----------------------------

Selection Number:  [2       ]          Selection Type:  [P]

Selection
Text:      [display output file from mailing labels report              ]

Selection
Action:    [cat mail.out                                                ]
```

The two actions of running the report and displaying the output file are now entered as details of the *mailinglabels* script menu. When the user selects option 3 on the **Reports** submenu, the *mailinglabels* script menu is selected, the mailing labels report is run, and the output file is displayed on the screen.

Use the Selection Type S when you want to run more than one action for a single menu item.

List of Appendixes

Appendix A. The *stores* Database

Appendix B. Environment Variables

Appendix C. INFORMIX-SQL Reserved Words

Appendix D. The *dbschema* Utility

Appendix E. Examples

Appendix F. ASCII Chart

Appendix G. The *bcheck* Utility

Appendix H. The *sqlconv* Utility

Appendix I. The *dbupdate* Utility

Appendix J. The *dbload* Utility

Appendix K. System Catalogs

Appendix L. Complex Outer Joins

Appendix M. Accessing Programs from the Operating System

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.

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.

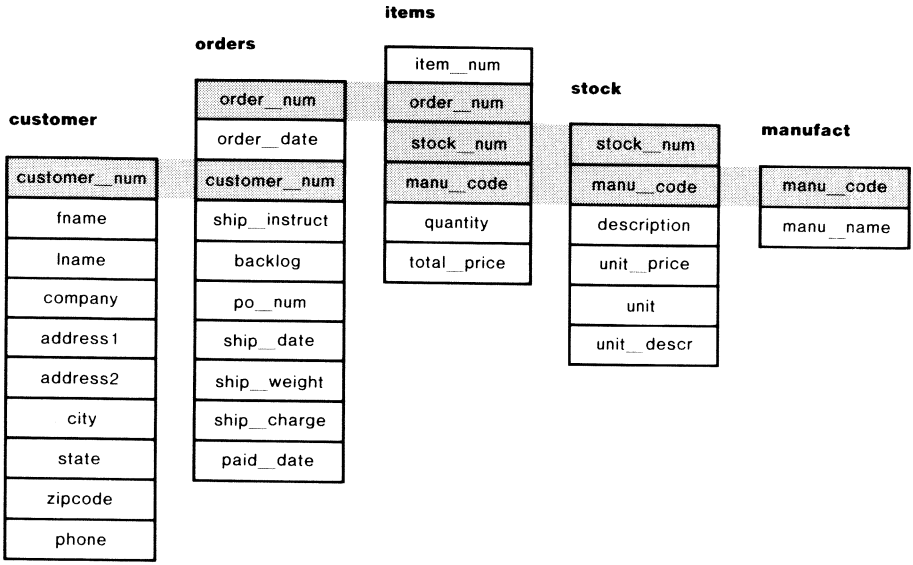
manufact

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 table (detail)

stock_num	manu_code	description
1	HRO	baseball glove
1	HSK	baseball glove
1	SMT	baseball glove
2	HRO	baseball

manufact table (detail)

manu_code	manu_name
NRG	Norge
HSK	Husky
HRO	Hero

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	fname	lname	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
add
after
all
allowing
alter
and
any
as
asc
attribute
attributes
autonext
average
avg
before
begin
bell
between
bottom
by
char
check
clipped
close
column
columns
comment
comments
commit
composites
connect
continue
count
create
database
date
day
dba
decimal
declare
default
define
delete
delimiters
desc
display
displayonly
distinct
do
downshift
drop
editadd
editupdate
else
end
every
exclusive
exists
exitnow
fetch
first
float
for
foreach
form
format
from
grant
group
having
header
headings
help
if
in
include
index
info
input
insert
instructions
integer
into
is
joining
last
left
length
let
like
line
lineno
lines
load
lock
log
lookup
margin

master
matches
max
mdy
min
mode
modify
money
month
need
nextfield
noentry
not
noupdate
null
of
on
option
or
order
outer
output
page
pageno
param
pause
percent
picture
pipe
print
printer
privileges
prompt
public
query
queryclear
remove
rename
repair
report
required
resource
reverse
revoke
right
rollback
rollforward
row
rowid
screen
select
serial
set
share
skip
smallfloat
smallint
some
space
spaces
start
statistics
status
step
sum
synonym
systables
table
tables
temp
then
time
to
today
top
total
trailer
type
union
unique
unload
unlock
update
upshift
using
values
variable
verify
view
weekday
where
while
with
without
work
year
zerofill

Appendix D

The *dbschema* Utility

The *dbschema* Utility

The Database Administrator (DBA) can use the **dbschema** utility to quickly produce an **RDSQL** command file containing the CREATE TABLE, CREATE INDEX, and CREATE VIEW statements required to replicate an entire database or a selected table. In addition, **dbschema** can produce all CREATE SYNONYM and GRANT statements in effect for the database or a selected table or view.

By default, **dbschema** produces all CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SYNONYM, and GRANT statements in effect for the entire database. By including the appropriate command line option, you can limit the output to a selected table or view, or produce synonyms and permissions for a particular user.

The syntax for the **dbschema** utility appears below:

---

dbschema [**-t** *tabname*] [**-s** *sname*] [**-p** *pname*] **-d** *database* [**-o** *filename*]

-t *tabname* identifies *tabname* as the table or view for which **dbschema** outputs the CREATE TABLE and CREATE INDEX statements or the CREATE VIEW statement. If *tabname* is all, the **RDSQL** statements for all database tables and views are output.

-s *sname* identifies *sname* as the user for which **dbschema** outputs the CREATE SYNONYM statements. If *sname* is all, the **RDSQL** statements for all users are output. If you include the **-t** option, **dbschema** produces the CREATE SYNONYM statements only for the indicated *table-name*.

-p *pname* identifies *pname* as the user for which **dbschema** outputs permissions. If *pname* is

all, the RDSQL statements necessary to grant permissions to all users are output. If you include the **-t** option, **dbschema** produces the GRANT statements only for the indicated *table-name*.

-d *database* is the name of the database.

-o *filename* is the name of the file receiving the RDSQL statements. If *filename* is not provided, **dbschema** writes to the terminal screen.

Notes

1. The following default command line produces the RDSQL statements necessary to replicate an entire database:

   ```
   dbschema -d database [-o filename]
   ```

2. When you include the **-t** option, **dbschema** produces RDSQL statements only for the indicated *table-name*. **dbschema** uses the *table-name* to filter the output. If you request CREATE SYNONYM and GRANT statements, these are provided only for *table-name*.

3. Most statements appear with comments. Depending upon the information requested, **dbschema** indicates the owner of the table, owner of the synonym, or grantor of a permission.

4. All SERIAL fields included in CREATE TABLE statements output by **dbschema** have a starting value of one, regardless of their original starting value.

5. The *database* must exist in your current directory or a directory cited in your DBPATH environment variable.

Examples

The following statement outputs the RDSQL statements relating to the **customer** table in the **stores** database to the **c_all** file.

```
dbschema -t customer -s nancy -p bettyjo -d stores -o c_a
```

The output consists of the following statements:

- The CREATE TABLE and CREATE INDEX statements replicating the **customer** table
- All CREATE SYNONYM statements executed by the user *nancy* on the **customer** table
- All permissions granted to the user *bettyjo* on the **customer** table

The output from the **dbschema** statement follows:

```
{enid is owner of table customer}
create table customer
    (
    custnum serial not null,
    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)
    );


{enid is owner of index c_num_ix}
create unique index c_num_ix on customer (custnum)

{enid is owner of index zip_ix}
create index zip_ix on customer (zipcode);

revoke all on customer from public;

{nancy is owner of synonym cust}
create synonym cust for customer;

{felix is the grantor}
grant all on customer to bettyjo;
```

The next **dbschema** statement outputs the **RDSQL** statements relating to all tables in the database to the **s_out** file.

```
dbschema -t all -s felix -p felix -d stores s_out
```

The output consists of the following statements:

- The CREATE TABLE, CREATE VIEW, and CREATE INDEX statements that replicate all tables, views, and indexes in the **stores** database
- All CREATE SYNONYM statements executed by the user *felix*
- All permissions granted to the user *felix*

Appendix E

Examples

Sample PERFORM Form Specifications

The *customer* Specification

```
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
```

The *orderform* Specification

```
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]
                   [manu_name       ]
                                        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
```

The *sample* Specification

```
database stores

screen
{
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════════════════════════════════════
CUSTOMER INFORMATION:

Customer Number: [c1          ]

        Company: [c4                    ]
     First Name: [c2          ]              Last Name: [c3              ]

        Address: [c5                    ]
                 [c6                    ]

           City: [c7              ] State: [c8]   Zip: [c9    ]
      Telephone: [c10             ]

═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════════════════════════════════════
}

screen

{
═══════════════════════════════════════════════════════════════════════
CUSTOMER NUMBER: [c1          ]          COMPANY: [c4                    ]

ORDER INFORMATION:
     Order Number: [o11          ]      Order Date: [o12           ]

     Stock Number: [i13          ]    Manufacturer: [i16]
      Description: [s14                            [m17             ]
             Unit: [s16                      ]
                                        Quantity: [i18          ]
                                       Unitprice: [s15                ]
                                     Total Price: [i19                ]

SHIPPING INFORMATION:
 Customer P.O.: [o20          ]        Ship Charge: [d1          ]

       Backlog: [a]           Total Order Amount: [d2          ]
    Ship Date: [o21          ]
    Date Paid: [o22          ]
  Instructions: [o23                                  ]

}
end
```

```
tables
customer items stock
orders manufact

attributes
c1 = *customer.customer_num
   = orders.customer_num;
c2 = fname,
       comments = "Please enter initial if available";
c3 = lname;
c4 = company, reverse;
c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
     include = ("CA","OR","NV","WA"),
     default = "CA" ;
c9 =  zipcode, autonext;
c10 = phone, picture = "###-###-####x####";
o11 = *orders.order_num = items.order_num;
o12 = order_date, default = today, format = "mm/dd/yyyy";
i13 = items.stock_num;
    = *stock.stock_num, noentry, noupdate, queryclear;
i16 = items.manu_code, lookup m17 = manufact.manu_name
   joining *manufact.manu_code, upshift, autonext;
    = *stock.manu_code, noentry, noupdate,
          upshift, autonext, queryclear;
s14 = stock.description, noentry, noupdate;
s16 = stock.unit_descr, noentry, noupdate;
s15 = stock.unit_price, noentry, noupdate;
i18 = items.quantity, include = (1 to 50),
comments = "Acceptable values are 1 through 50" ;
i19 = items.total_price;
o20 = po_num, required,
comments = "If no P.O. Number enter name of caller" ;
a   = backlog, autonext;
o21 = ship_date, default = today, format = "mm/dd/yyyy";
o22 = paid_date, format = "mm/dd/yyyy";
o23 = ship_instruct;
d1 =  displayonly type money;
d2 =  displayonly type money;

instructions

customer master of orders;
orders master of items;
composites <items.stock_num, items.manu_code>
      *<stock.stock_num, stock.manu_code>

before editadd editupdate of orders
   nextfield = o20

before editadd editupdate of items
   nextfield = i13

after editadd editupdate of quantity
   let i19 = i18 * s15
   nextfield = o11

after add update query of items

   if (total of i19) <= 100 then
      let d1 = 7.50
      else
      let d1 = (total of i19) * .04

   let d2 = (total of i19) + d1

after display of orders

   let d1 = 0
   let d2 = 0

end
```

Sample ACE Report Specifications

The *clist1* Specification

```
{ File: clist1.ace - Customer List Specification 1}

database
        stores
end

output
        left margin 2
end

select
        customer_num,
        fname,
        lname,
        company,
        city,
        state,
        zipcode,
        phone
    from
        customer
    order by
        city
end

format

first page header
    print column 32, "CUSTOMER LIST"
    print column 32, "-------------"
    skip 2 lines
    print "NUMBER",
        column 9, "NAME",
        column 32, "LOCATION",
        column 54, "ZIP",
        column 62, "PHONE"
    skip 1 line

page header
    print "NUMBER",
        column 9, "NAME",
        column 32, "LOCATION",
        column 54, "ZIP",
        column 62, "PHONE"
    skip 1 line

on every row
    print customer_num using "####",
        column 9, fname clipped, 1 space, lname clipped,
        column 32, city clipped, ", " , state,
        column 54, zipcode,
        column 62, phone

on last row
    skip 1 line
    print  "TOTAL NUMBER OF CUSTOMERS:",
        column 30, count using "##"
end
```

The *clist2* Specification

```
{ File:  clist2.ace -  Customer List Specification 2 }

database
        stores
end

define
    variable thisstate char(2)
end

input
    prompt for thisstate using
        "Enter state (use UPPER CASE) for which you wish a customer list:  "
end

output
    left margin 0
end

select
        customer_num,
        fname,
        lname,
        company,
        city,
        state,
        zipcode,
        phone
    from
        customer
    where
        state matches $thisstate
    order by
        zipcode,
        lname
end

format

    first page header
    print column 32, "CUSTOMER LIST"
    print column 32, "-------------"
    skip 2 lines
    print "Listings for the State of ", thisstate
    skip 2 lines
    print "NUMBER",
        column 9, "NAME",
        column 32, "LOCATION",
        column 54, "ZIP",
        column 62, "PHONE"
    skip 1 line

    page header
    print "NUMBER",
        column 9, "NAME",
        column 32, "LOCATION",
        column 54, "ZIP",
        column 62, "PHONE"
    skip 1 line

    on every row
    print customer_num using "####",
        column 9, fname clipped, 1 space, lname clipped,
        column 32, city clipped, ", " , state,
        column 54, zipcode,
        column 62, phone

  on last row
    skip 2 lines
    print "Number of customers in ",thisstate, " is ", count using "<<<<&"
  end
```

The *mail1* Specification

```
{file   mail1.ace

Mailing Label Specification - 1 }

{Customized report to print mailing labels.
The report will be sorted by zip code and
last name.  The use of PRINT commands in
the FORMAT section allow specific columns
to be printed on a line.
Blank lines will appear where data is
absent, and spaces appear next to
city field as not clipped as in
mail2.ace specification
}

database
stores
end

select *
from customer
order by zipcode, lname
end

format
  on every row
     print fname, lname
     print company
     print address1
     print address2
     print city, ", " , state, 2 spaces, zipcode
     skip 2 lines
end
```

The *mail2* Specification

```
{mail2.ace    file

Mailing Label Specification - 2 }

{This improved report has an OUTPUT section added,
and uses nested IF statements.  }

database
stores
end

output
  top margin 0
  bottom margin 0
  left margin 0
  page length 9
  report to "labels"
end

select
fname, lname,
company,
address1,
address2,
city, state, zipcode
from customer
order by zipcode, lname
end

format
  on every row
    if (city is not null) and
      (state is not null) then
    begin
       print fname clipped, 1 space, lname
       print company
       print address1
       if (address2 is not null) then
          print address2
       print city clipped, ", " , state,
          2 spaces, zipcode
       skip to top of page
    end
end
```

The *mail3* Specification

```
{file   mail3.ace

Mailing Label Specification - 3 }

{This report prints 1-3 mailing labels across a page.
It stores the labels in character strings (array1,
array2, and array3) as it reads each row, and prints
the labels when it has read the proper number of rows.
At run time, you specify the number of labels (1-3)
that you want ACE to print across the page.  }

database
stores
end


define
    variable name      char(75)              {holds first and last names}
    variable cstzp     char(75)              {holds city, state, and zip}
    variable array1    char(80)              {Array for name line}
    variable array2    char(80)              {Array for street line}
    variable array3    char(80)              {Array for city, state, and zipcode}
    variable start     smallint              {start of current label in array}
    variable finish    smallint              {end of current label in array}
    variable l_size    smallint              {label width}
    variable white     smallint              {spaces between each label}
    variable count1    smallint              {number of labels across page}
    variable i         smallint              {label counter}
end


input
    prompt for count1 using "Number of labels across page? [1-3] "
end

output
    top margin 0
    bottom margin 0
    left margin 0
    report to "labels.out"
end


select
        *
    from
        customer
    order by
        zipcode
end
```

```
format
   first page header                          {Nothing is displayed in this
                                               control block.  It just
                                               initializes variables that are
                                               used in the ON EVERY ROW
                                               control block.}
      let i = 1                               {Initialize label counter.}
      let l_size = 72/count1                  {Determine label width (allow
                                               eight spaces total between labels).}
      let white = 8/count1                    {Divide the eight spaces between
                                               the number of labels across the page.}

   on every row
      let name = fname clipped, 1 space, lname
      let cstzp = city clipped,
                  ", ",
                  state,
                  2 spaces, zipcode
      let finish = (i * l_size) + white
                                              {This section assigns names, }
      let start = finish - l_size
                                              {addresses, and zip codes to }
      let array1[start, finish] = name
                                              {arrays 1, 2, 3 until        }
      let array2[start, finish] = address1
                                              {i = the number of labels    }
      let array3[start, finish] = cstzp
                                              {across a page.              }

      if i = count1 then
         begin
            print array1 clipped              {Print the stored addresses.}
            print array2 clipped              {Use clipped to remove trailing}
            print array3 clipped              {spaces for quicker printing.}

            skip 1 line

            let array1 = " "                  {Reset the arrays to spaces.}
            let array2 = " "
            let array3 = " "

            let i  = 1
         end
      else
         let i = i + 1

   on last row
      if i > 1 then                           {Print the last set of addresses}
         begin                                {if there were any left.}
            print array1 clipped
            print array2 clipped
            print array3 clipped
         end
end
```

The *ord1* Specification

```
{file   ord1.ace

Order Specification - 1 }

database
stores
end

output
  report to "ordlist1"
end

select

  orders.order_num ordnum,
  order_date, customer_num,
  po_num, ship_date, ship_charge,
  paid_date,

  items.order_num, stock_num, manu_code,
  quantity, total_price

  from orders, items

  where orders.order_num = items.order_num

  order by ordnum
end

format

before group of ordnum
  print "Order number:  ", ordnum using "#####",
    " for customer number:  ", customer_num
    using "#####"
  print "Customer P.O. :  ", po_num,
    "  Date ordered:  ", order_date
  skip 1 line
  print "Stockno", column 20,
    "Mfcode", column 28, "Qty", column 38, "Price"

on every row
  print stock_num using "###", column 20,
    manu_code, column 28, quantity using "###",
    column 38, total_price using "$$$,$$$.&&"

after group of ordnum
  skip 1 line
  print 5 spaces, "Total amount for the order:  ",
    group total of total_price using "$$,$$$,$$$.&&"
  skip 3 lines


end
```

The *ord2* Specification

```
{file   ord2.ace

Order Specification - 2 }

database
stores
end

output
   left margin 0
   report to "ordlist2"
end

select

   customer.customer_num custnum, fname,
     lname, company,

   orders.order_num ordnum, order_date,
     orders.customer_num, po_num, ship_date,
     ship_charge, paid_date,

   items.order_num, items.stock_num snum,
     items.manu_code, quantity, total_price,

   stock.stock_num, stock.manu_code,
     description, unit_price

 from  customer, orders, items, stock

 where customer.customer_num = orders.customer_num
   and orders.order_num = items.order_num
   and items.stock_num = stock.stock_num
   and items.manu_code = stock.manu_code

 order by custnum,  ordnum, snum

end
```

```
format

before group of custnum
  print "Orders for:  ", fname clipped, 1 space,
  lname print 13 spaces, company
  skip 1 line

before group of ordnum
  print "Order number:  ", ordnum using "#####"
  print "Customer P.O. :  ", po_num,
    "  Date ordered:  ", order_date
  skip 1 line
  print "Stockno", column 10, "Mfcode", column 18,
    "Description", column 38, "Qty", column 43,
    "Unit price" , column 55, "Total for item"

on every row
  print snum using "###", column 10, manu_code,
    column 18, description clipped, column 38,
    quantity using "###", column 43,
    unit_price using "$$$$.&&", column 55,
    total_price using "$$,$$$,$$$.&&"

after group of ordnum

  skip 1 line
  print 4 spaces,
    "Shipping charges for the order:  ",
    ship_charge using "$$$$.&&"
  skip 1 line

  print 5 spaces, "Total amount for the order:  ",
    ship_charge + group total of
    total_price using "$$,$$$,$$$.&&"
  skip 3 lines

after group of custnum
  skip 2 lines

end
```

The *ord3* Specification

```
{file   ord3.ace

Order Specification - 3 }

database
stores
end

define
  variable begin_date date
  variable end_date date
end

input
  prompt for begin_date using
    "Enter beginning date for report:  "
  prompt for end_date using
    "Enter ending date for report:  "
end

output
  left margin 0
  report to "ordlist3"
end

select

  customer.customer_num, fname, lname, company,

  orders.order_num ordnum, orders.customer_num,
  order_date, month(order_date) months,
  day(order_date) days, year(order_date) years,

  items.order_num, quantity, total_price

  from  customer, orders, items

  where customer.customer_num = orders.customer_num
    and orders.order_num = items.order_num and
    order_date between $begin_date and $end_date

  order by years, months, days, company, ordnum
end
```

```
format

first page header
  print column 10, "=============================================================="
  print column 10, "                          DAILY ORDER REPORT"
  print column 10, "=============================================================="
skip 1 line
  print column 15, "FROM:  ", begin_date
    using "mm/dd/yy",
    column 35, "TO:  ", end_date
    using "mm/dd/yy"
  print column 15, "Report run date:  ",
    today using "mmm dd, yyyy"
  skip 2 lines
  print column 2, "ORDER DATE", column 15,
    "COMPANY", column 35, "NAME",
    column 57, "NUMBER", column 65, "AMOUNT"

before group of days
  skip 2 lines

after group of ordnum
  print column 2, order_date, column 15,
    company clipped, column 35, fname clipped,
    1 space, lname clipped, column 55,
    ordnum using "####", column 60,
    group total of total_price using "$$,$$$,$$$.&&"

after group of days
  skip 1 line
  print column 21, "Total amount ordered for the day:  ",
    group total of total_price using "$$$$,$$$,$$$.&&"
  skip 1 line
  print column 15, "=============================================================="

on last row
  skip 1 line
  print column 15, "=============================================================="
  skip 2 lines
  print "Total Amount of orders:  ", total of
    total_price using "$$$$,$$$,$$$.&&"

page trailer
   print column 28, pageno using "page <<<<"

end
```

Appendix F

ASCII Chart

Num	Char	Num	Char	Num	Char
0	^@	43	+	86	V
1	^A	44	,	87	W
2	^B	45	-	88	X
3	^C	46	.	89	Y
4	^D	47	/	90	Z
5	^E	48	0	91	[
6	^F	49	1	92	\
7	^G	50	2	93	]
8	^H	51	3	94	^
9	^I	52	4	95	_
10	^J	53	5	96	`
11	^K	54	6	97	a
12	^L	55	7	98	b
13	^M	56	8	99	c
14	^N	57	9	100	d
15	^O	58	:	101	e
16	^P	59	;	102	f
17	^Q	60	<	103	g
18	^R	61	=	104	h
19	^S	62	>	105	i
20	^T	63	?	106	j
21	^U	64	@	107	k
22	^V	65	A	108	l
23	^W	66	B	109	m
24	^X	67	C	110	n
25	^Y	68	D	111	o
26	^Z	69	E	112	p
27	esc	70	F	113	q
28	^\	71	G	114	r
29	^]	72	H	115	s
30	^^	73	I	116	t
31	^_	74	J	117	u
32		75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	|
39	'	82	R	125	}
40	(	83	S	126	~
41	)	84	T	127	del
42	*	85	U		

^x = CONTROL-X

Appendix G

The *bcheck* Utility

The *bcheck* Utility

The **bcheck** program is a C-ISAM utility program that checks and repairs C-ISAM index files. It is distributed with INFORMIX-SQL and is called when you run the CHECK TABLE and REPAIR TABLE RDSQL statements.

bcheck compares an index (**.idx**) file to a data (**.dat**) file to see whether the two are consistent. If they are not, **bcheck** asks whether you want to delete and rebuild the corrupted indexes.

When running **bcheck** from the operating system command line, you must specify the table name used by the INFORMIX-SQL system catalogs. You must specify the full pathname of the table if you are not in the **.dbs** directory. For example, the **customer** table in the **stores** database is identified in the system catalogs as **custome100** (on UNIX systems) or **custo100** (on DOS systems) and not as **customer.**

You can list the contents of the database directory to determine the appropriate table name.

In the following example, **bcheck** is run from the command line on the **customer** file and finds no errors.

```
bcheck -n custome100

BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: custome100

Checking dictionary and file sizes.
Index file node size = 1024
Current C_ISAM index file node size = 1024
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    0 index node(s) used -- 1 index b-tree level(s) used
Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used
Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used
Checking data record and index node free lists.
4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

For each index, **bcheck** prints a group of up to eight numbers. These numbers indicate the position of the key in each record.

You can also use **bcheck** with the following options:

-i Check index file only
-l List entries in B+ trees
-n Answer no to all questions
-y Answer yes to all questions
-q Suppress printing of the program banner
-s Resize the index file node size

The **bcheck** command syntax is as follows:

bcheck -[**i** | **l** | **y** | **n** | **q** | **s**] *file-name*

Unless you use the **-n** or **-y** option, **bcheck** is interactive, waiting for you to respond to each error it finds.

Use the **-y** option with caution. Do not run **bcheck** using the **-y** option if you are checking the files for the first time.

The CHECK TABLE statement of **RDSQL** runs **bcheck** with the **-n** option. The REPAIR TABLE statement of **RDSQL** runs **bcheck** with the **-y** option.

Here is a sample run in which **bcheck** finds errors. The **-n** option is selected, so that each question **bcheck** asks is automatically answered "no."

```
BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: custome100

Checking dictionary and file sizes.
Index file node size = 1024
Current C_ISAM index file node size = 1024
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    0 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Checking data record and index node free lists.

ERROR:  3 missing data record(s)
Fix data record free list  ? no

4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

Since **bcheck** finds errors, you must delete and rebuild the corrupted indexes. The **-y** option is used to answer "yes" to all questions asked by **bcheck**:

```
BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: custome100

Checking dictionary and file sizes.
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes
Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes

Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes

Checking data record and index node free lists.

ERROR:  3 missing data record(s)
Fix data record free list  ? yes

Recreate data record free list
Recreate index 3
Recreate index 2
Recreate index 1

4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

Appendix H

The *sqlconv* Utility

The *sqlconv* Utility

The **sqlconv** utility is provided for the Database Administrator who wants to use INFORMIX-SQL with an existing INFORMIX database (Version 3.0 or higher). From the INFORMIX database, **sqlconv** aids a user to create a new RDSQL -compatible database. It leaves the old database intact.

This appendix discusses the steps necessary to convert an INFORMIX database to an RDSQL -compatible database. The new database can be used with INFORMIX-SQL on either DOS or UNIX systems.

This appendix describes two methods you can use to accomplish the conversion. The method you use depends on the disk space constraints of your system.

Caution! sqlconv will not convert INFORMIX database permissions. You must grant new permissions on tables and fields after your new RDSQL database has been created and loaded.

RDSQL reserved words are not the same as INFORMIX reserved words. If you receive a syntax error while running your new CREATE scripts, make sure your TABLE and FIELD names are not among the new reserved words. For a list of reserved words, see the "Reserved Words" appendix.

Make sure all INFORMIX composite fields are indexed. RDSQL takes no action on unindexed composite fields. Indexed composite fields will have composite indexes created for them.

If you have used the LOCATION option to spread an INFORMIX database across a number of directories, converting the database using **sqlconv** places all of these files in the new **.dbs** directory.

You cannot specify a new starting number for a serial field.

If There Is No Shortage of Disk Space

You can convert an entire **INFORMIX** database to an **INFORMIX-SQL** database at one time if there is no shortage of disk space.

To convert an entire database at once, you must have available on the disk at least three times the amount of space required by all the tables in the database plus additional space for the **INFORMIX-SQL** system files. The following steps outline the process of converting an entire database at once:

1. Make certain that **INFORMIX** and **INFORMIX-SQL** are included in your search path.

2. Set the INFORMIXDIR environment variable to point to the **INFORMIX-SQL** directory.

3. Change your current directory to the directory that contains your **INFORMIX** database.

4. Create a backup copy of the **INFORMIX** database.

5. Enter

   sqlconv *databasename*

   where *databasename* is the name of the **INFORMIX** database you want to convert. Do not include a filename extension. This command generates an **INFORMER** script file (indicated by a **.uld** extension) and an **INFORMIX-SQL** script file (indicated by a **.sql** extension).

6. Enter

   informer *databasename databasename*.**uld**

   This command unloads the database files (in ASCII format) to *unload* files (indicated by a **.unl** extension) for each table in the database.

7. Enter

   isql - *databasename*.**sql**

   This command re-creates the database, tables, and indexes in **RDSQL** format and loads the data from the **.unl** files (generated by **informer**) into the appropriate tables.

8. The final step in the conversion procedure is to remove all the old database files, the **.uld** files, the **.sql** files, and the **.unl** files, from your directory. Do not remove your forms and reports. You can update these later.

If There Is a Shortage of Disk Space

The following method is more economical in terms of disk space, but it is a more involved and time-consuming process than the method discussed in the previous section. Use this method if you do not have at least three times the amount of space required by the database.

The following steps outline the conversion of the sample **INFORMIX** database, **payroll**, to an **INFORMIX-SQL** database:

1. Make certain that **INFORMIX** and **INFORMIX-SQL** are included in your search path.

2. Set the INFORMIXDIR environment variable to point to the **INFORMIX-SQL** directory.

3. Change your current directory to the directory that contains your **INFORMIX** database.

4. Create a backup copy of the INFORMIX database.

5. Enter

```
sqlconv payroll
```

   Do not include a filename extension. This command generates the **.uld** and **.sql** scripts necessary to convert the database.

6. Create a copy of the **payroll.sql** file, and make sure you include the **.sql** extension to the filename of the new file. In this instance, we have named this file **createp.sql**.

7. Edit the **createp.sql** file using your system editor, and remove all the **load** statements from the file. The CREATE DATABASE, CREATE TABLE, and CREATE INDEX statements should be the only statements remaining in the file. Exit the file.

8. Enter

```
isql - createp.sql
```

   This command creates the INFORMIX-SQL database and its tables and indexes, but it does not load the data.

9. On UNIX systems, enter

```
cat payroll.uld
```

   On DOS systems, enter

```
type payroll.uld
```

   The statements in the **payroll.uld** file outline the first steps of the conversion operation. You must execute each of these statements separately. You may find it helpful to print a copy of the **payroll.uld** file for easy reference.

10. Enter

    ```
    informer payroll
    ```

    At the **INFORMER** prompt, enter the first line of the **payroll.uld** file exactly as it appears. This creates the unload file for the first table. Exit **INFORMER**.

11. On UNIX systems, locate the first load statement in the **payroll.sql** file by entering

    ```
    cat payroll.sql
    ```

    On DOS systems, locate the first load statement in the **payroll.sql** file by entering

    ```
    type payroll.sql
    ```

12. Enter

    ```
    isql payroll -
    ```

    At the > prompt, enter the first load statement that appears in the **payroll.sql** file. Exit to the operating system.

13. Before you can perform the same operations on any other database tables, you must free additional disk space by erasing the **INFORMIX** versions of the recently created **INFORMIX-SQL** files. To erase these files, enter

    ```
    dbstatus payroll
    ```

14. At the **dbstatus** prompt, enter

    erase file *filename*

    where *filename* is the name of the file referred to in Step 12. Exit **dbstatus.**

15. On UNIX systems, erase the unload file by entering from the command line

```
rm filename.unl
```

On DOS systems, erase the unload file by entering from the command line

```
erase filename.unl
```

16. Continue to unload each table, one at a time, from the INFORMIX database and then load it into the INFORMIX-SQL database. Do this by repeating Steps 10 through 15 until all load statements in the **payroll.sql** file have been executed. This operation loads the actual data into the newly created database.

17. When all tables in the database have been converted, erase the **.sql** files and the **.uld** files, and drop the INFORMIX database.

18. Check to see that your database has been successfully converted to an INFORMIX-SQL database, and that your data has been successfully loaded into the tables, before you drop the backup copy of the INFORMIX database.

Caution! You cannot rerun sqlconv after you have erased a table. The new scripts generated by the command do not contain the information necessary to convert the table.

Appendix I

The *dbupdate* Utility

The *dbupdate* Utility

Several significant changes have occurred in RDSQL in the transition from Version 1 to Version 2. These changes include expanding the number of system catalogs and slightly changing their content (see Appendix K), and introducing NULL values.

Version 2.1 of INFORMIX-SQL and INFORMIX-ESQL/C and Version 1.1 of INFORMIX-4GL use the Version 2 of RDSQL. You cannot use databases constructed under Version 1 with the Version 2.1 products or with INFORMIX-4GL unless you modify the old databases with the **dbupdate** utility.

Using dbupdate

To convert your old databases to an RDSQL Version 2 database, execute the following command:

---

dbupdate [**-b** | **-n**] *old-database-name new-database-name*

---

dbupdate creates a new RDSQL -compatible database in the current directory, with the name *new-database-name*, and copies the data from the old system catalogs to the new system catalogs, making the appropriate changes.

If you do not use the **–b** or **–n** option, **dbupdate** converts the value of all CHAR type columns with blank data to NULL and, for each numeric column, asks you whether it should convert zero values to NULL values.

The **–b** option causes **dbupdate** to leave blank data in CHAR columns as blanks. The **–n** option alters the system catalogs to define all columns as NOT NULL and does not touch the data files. The **–n** option includes the **–b** option.

In addition to these changes, **dbupdate** corrects a bug in the representation of negative DECIMAL values.

When **dbupdate** finishes, you have two database directories (the new and the old) with two separate system catalogs, but the data and index files are shared (linked). To complete the update, you should remove the old database directory.

On DOS systems, the **dbupdate** utility calls the *sqlexec.sqc* program to convert a Version 1.1 database to a Version 2.0 or 2.1 database. Once you have updated your database, the *sqlexec.sqc* program is no longer needed. Because the *sqlexec.sqc* program is large, you can delete it once the database conversion is complete.

Caution! On DOS systems, executing the **dbupdate** command loads the database agent. Be certain that the database agent is not already loaded in memory when executing the **dbupdate** command.

To find out if the database agent is loaded, enter **set** from the command line. This displays a list of your DOS environment variables. If the database agent is loaded, the variable **sqlcaddr** is listed. If **sqlcaddr** is listed, you must unload the database agent using the **exit** command before you run **dbupdate.**

No NULL Databases

You may want to avoid dealing with NULL values and their three-valued logic. You can do this if you carefully adhere to the following rules:

- When converting the **RDSQL** Version 1 database to an **RDSQL** Version 2 database, select the **–n** option of **dbupdate**.
- When creating new tables, define all columns as NOT NULL.
- In all form specification files, add the WITHOUT NULL INPUT clause in the DATABASE section.
- In form specification files, specify all **displayonly** fields as NOT NULL.

The last two rules mean you must recompile all your old form specification files.

Appendix J

The *dbload* Utility

The *dbload* Utility

The **dbload** utility provides a method for transferring information (raw data existing in ASCII form) from an operating-system file into an **RDSQL** version 2.1 database. It allows for the easy and efficient transfer of database files created for other **RDSQL** databases—or created on an entirely different database system—to your **RDSQL** version 2.1 database. Error logging provides diagnostic information about rows that are not successfully loaded.

When running **dbload**, you have the option of specifying the point at which each *N* number of rows is committed to the database—or of aborting the entire process and making no changes to the database.

These are the primary features of **dbload**:

- Data from one or more system files can be loaded into one or more database tables.
- Rows that cannot be loaded into the database are trapped and placed (with diagnostic information) into a user-specified file.
- Loading continues until a user-specified total number of rows cannot be entered into the database.
- The loading process can be optionally terminated without making changes to the existing database.
- Loading can begin at a specified line in the system file.
- Selected fields or columns can be loaded from the system file and entered in specified columns of a table.
- Both fixed- and variable-length data files can be loaded.
- NULL values can be indicated, and a different indicator can be used with each field.

- Constants (values not in the system file) can be entered into the database file.

To use **dbload**, first create a command file that tells **dbload** what actions you want carried out.

This command file

- Describes the form of the data in the system files
- Contains one or more RDSQL INSERT INTO statements indicating how this data is to be placed into the selected database files

You invoke **dbload** from the operating system command line by indicating (at a minimum) the name of the **dbload** program and an option. If you provide no additional information on the command line, **dbload** prompts you for the information (database name, command filename, and error-log filename) it must have to run.

The syntax of the basic **dbload** command is

---

dbload { **-d** *dbname* | **-c** *comfile* | **-l** *errlog* } ...

-d *dbname* is the name of the RDSQL database receiving the new information.

-c *comfile* is the name of the command file containing the **dbload** command statements.

-l *errlog* is the name of the file selected to receive the error-logging information.

dbload can be used with several options. Options must be placed on the command line. The following options are available:

-s The **-s** option instructs **dbload** to perform a syntax check on the command file only and load no data. The command file is displayed on the screen, and errors are indicated where they are found. You can redirect this output to a system file with the > redirect symbol.

You will normally want to run **dbload** with the **-s** option in advance of actually loading the data.

-n *num* **dbload** displays a message when each set of *num* rows is loaded into the database.

For databases with transactions, the **-n** *num* indicates the number of rows that must be successfully read by **dbload** before being committed to the database. Until this number is reached, no rows are committed to the database. The default is 100 rows.

-e *num* The **-e** *num* indicates the total number of bad rows that will be processed before **dbload** aborts the program. The default is 10 total rows.

For example, if *num* is set to 10, **dbload** aborts upon reaching the eleventh error. **dbload** aborts upon encountering the first bad row if *num* is set to 0.

-p The **-p** option instructs **dbload** to ask whether to commit rows to the database when the program reaches an abort point. The rows that are committed are those read since the last set of rows was committed. The default is to commit the rows to the database without asking for verification.

-i *num* The **-i** *num* option instructs **dbload** to ignore the first *num* number of rows in the system data file when executing your request. For example, this option is useful if the **dbload** program aborts after

line 240 of the system file data, and you want to begin loading with line 241. It is useful also if the system file contains header information preceding the data.

An example operating-system command line using all options (except **-s**) is

```
dbload -d stores -c comfile -l logfile
                 -n 50 -e 5 -p -i 20
```

In the example command line:

-d stores	stores is the name of the database
-c comfile	comfile is the name of the command file
-l logfile	logfile is the name of the error-log file
-n 50	**dbload** commits each set of 50 good rows to the database
-e 5	**dbload** allows 5 bad rows before aborting
-p	**dbload** prompts for instructions about committing rows read, but not yet committed to the database, upon reaching the abort point
-i 20	**dbload** ignores the first 20 lines of the data file

To abort the **dbload** program, press the program interrupt key (DEL on UNIX systems, CTRL-C on DOS systems). Upon interrupt, rows read but not committed to the database are discarded.

Suggestion: The speed with which the **dbload** utility loads data is greatly affected by the presence of indexes. Before running **dbload** you may want to drop any indexes on the tables receiving the data. Then add new indexes once the program has finished.

The format of an example command file follows. A discussion of each line in this file follows the example.

```
FILE datafile1 (fld1 1 - 10 : 13 : 5 - 22  NULL = "str1" ,
                fld2 10 - 21 : 28 - 32 ,
                fld3 8 - 10 : 33 - 50 : 29 - 33  NULL = "str2" ,
                                .
                                .
                                .
                fldN 9 : 16 - 19  NULL = "string") ;

INSERT INTO tab1 (col1, col2, col9, . . . . , colN) ;

INSERT INTO tab2
     VALUES  (fld1, fld3, "kevin", . . . . , fldN) ;

INSERT INTO tab3 ;              {no column or values list provided}

FILE "datafile.2" DELIMITER "|" Num ;     {variable length fields}

INSERT INTO tab1
     VALUES  (f01, f02, "kevin", "234", . . . . , f0N) ;

INSERT INTO tab4 ;
```

The preceding command file contains seven statements. Each statement is described below.

```
FILE datafile1 (fld1 1 - 10 : 13 : 5 - 22  NULL = "str1" ,
```

datafile1 is the name of the operating system data file. *fld1*, *fld2*, *fld3*, through *fldN* are the field names you select to identify each fixed-length data field.

In this example, *fld1* consists of the characters in positions 1 through 10, 13, and 5 through 22 of every row of *datafile1*. (All rows end with a NEWLINE character.) Character positions can be repeated in each field and across fields. Character positions 5 through 10 and 13 appear twice in *fld1*, and characters 10, 13, and 21 appear in fields *fld1* and *fld2*.

In *fld1* the NULL value is defined as *"str1."* **dbload** places a NULL value in the column wherever *"str1"* is encountered.

fld2 consists of the characters in positions 10 through 21 and 28 through 32.

fld3 consists of the characters in positions 8 through 10, 33 through 50, and 29 through 33. The NULL value in this field is defined as *"str2"*.

The field-definition process continues until the last field is reached. Here, *fldN* contains characters in positions 9 and 16 through 19. The NULL value is defined as *"string"*.

Characters and character sequences are separated within each field definition by a colon. Field definitions are separated from one another by a comma. The end of the set of field definitions is indicated with a right parenthesis and a semicolon. All statements end with a semicolon.

```
INSERT INTO tab1 (col1, col2, col9, . . . , colN) ;
```

A standard RDSQL INSERT statement follows. *col1*, *col2*, and so on are the actual database column names. Since no value list is provided, **dbload** takes the value list from the preceding FILE statement. **dbload** inserts *fld1* data into *col1*; *fld2* into *col2*; and *fld3* into *col9* of table *tab1*. (In this instance, columns 4 through 8 are skipped.) This process continues until *fldN* is inserted into *colN*.

```
INSERT INTO tab2
    VALUES (fld1, fld3, "kevin", . . . , fldN) ;
```

Since no column list is provided, **dbload** checks the system catalogs for the column list for table *tab2*. The values to be loaded into each column are specifically stated. Here, *fld1* will be loaded as column1; *fld3*, as column2; and the constant *"kevin,"* as column3. The process continues until *fldN* appears in the final column.

```
INSERT INTO tab3 ;      {no column or values list provided}
```

Since no column or value list is provided, **dbload** checks the system catalogs for the column list for table *tab3*, and the value list is taken from the prior FILE statement (appearing at the start of the command file).

In this example, the value *fld1* is placed in column1; *fld2* in column2; *fld3* in column3; and *fldN* in columnN.

Note: This statement requires that the field list in the FILE statement have a one-to-one correspondence with the columns in the system catalogs. If this correspondence is not present,

dbload will not load the rows. Instead, you will receive an error message on each row. **dbload** aborts the loading process when the total number of bad rows exceeds the user-specified limit (if indicated) or the default limit of 10 bad rows.

A comment {no column or values list provided} is enclosed in braces.

```
FILE "datafile.2" DELIMITER "|" Num ; {variable length fields}
```

Columns in *datafile.2* are of variable length. The DELIMITER clause in this statement tells **dbload** that the "|" character separates each column. The same delimiter must be used throughout the file and must appear within quotation marks. *Num* is the number of columns that make up one row in the data file. Fields are automatically assigned the names *f01*, *f02*, *f03*, and so on. Any field that contains no characters is assigned a NULL value (CHAR fields are padded with blanks).

Because the filename contains an extension (.2), the filename must be enclosed in quotes ("*datafile.2*").

```
INSERT INTO tab1
    VALUES (f01, f02, "kevin", "234", . . . , f0N) ;
```

Since no column list is provided, **dbload** checks the system catalogs for this information. The values to be loaded into each column of *tab1* are specifically stated. Here, *f01* will be loaded as column1; *f02*, as column2; the constant *"kevin,"* as column3; the value *"234"*, as column4; and so on, until the value *f0N* is placed in the last column.

You must referenence fields in a variable length data file with the letter "f" followed by a two digit number: *f01*, *f02*, *f10*, and so on. The format *fld01* or *f3* is incorrect.

```
INSERT INTO tab4 ;
```

Since no column or value list is provided, **dbload** checks the system catalogs for the column list for table *tab4*. The value list is taken from the previous FILE statement. The file list in the FILE statement must have a one-to-one correspondence with the columns in the system catalogs. In this instance, the value *f01* is placed in column1; *f02* in column2; *f3* in column3; and so on, until the value *f0N* is placed in columnN.

The complete operating-system command line syntax is

---

dbload { **-d** *dbname* | **-c** *cfile* | **-l** *lfile* } ...
[**-s**] [**-n** *num*] [**-e** *num*] [**-p**] [**-i** *num*]

-d *dbname* is the name of the database receiving the new information.

-c *cfile* is the name of the **dbload** command file.

-l *lfile* is the name of the error logging file.

-s instructs **dbload** only to check the syntax of the *cfile* statement—no data is loaded into the database.

-n *num* gives the number of rows (*num*) read by **dbload** before committing the rows to the database (default = 100 rows).

-e *num* is the total number of bad rows (*num*) that are processed before **dbload** aborts (default = 10 rows; **dbload** aborts on bad row 11.)

-p instructs **dbload** to ask whether to commit rows to the database when the program reaches an abort point. The rows that are committed are those read since the last set of rows was committed. The default is to commit the rows to the database without asking for verification.

-i *num* instructs **dbload** to ignore the first *num* number of rows in the data file.

You must include at least one of the above options. If you do not include the first three options, **dbload** prompts for the information (database name, command filename, and error logging filename) it must have.

Appendix K

System Catalogs

System Catalogs

Information about the database is maintained in the system catalogs. The system catalogs are

systables	describes database tables.
syscolumns	describes columns in tables.
sysindexes	describes indexes on columns.
systabauth	identifies table-level privileges.
syscolauth	identifies column-level privileges.
sysdepend	describes how views depend on tables.
syssynonyms	lists synonyms for tables.
sysusers	identifies database-level privileges.
sysviews	defines views.

The following listing gives a brief description of the system catalogs.

The SYSTABLES Catalog

Column Name	Type	Explanation
tabname	char(18)	name of table
owner	char(8)	owner of table
dirpath	char(64)	directory path for the table file
tabid	serial	internal table identifier
rowsize	smallint	row size
ncols	smallint	number of columns
nindexes	smallint	number of indexes
nrows	integer	number of rows
created	date	date created
version	integer	table version number
tabtype	char(1)	table type (T = table, V = view, L = transaction log)
audpath	char(64)	audit filename (full pathname)

Index Name	Type	Columns
tabname	unique	tabname, owner
tabid	unique	tabid

The SYSCOLUMNS Catalog

Column Name	Type	Explanation
colname	char(18)	column name
tabid	integer	table identifier
colno	smallint	column number
coltype	smallint	column type
collength	smallint	column length (physical)

Index Name	Type	Columns
column	unique	tabid, colno

The SYSINDEXES Catalog

Column Name	Type	Explanation
idxname	char(18)	index name
owner	char(8)	owner of index
tabid	integer	table identifier
idxtype	char(1)	index type (U = unique, D = dups)
clustered	char(1)	clustering
part1	smallint	column number
part2	smallint	column number
part3	smallint	column number
part4	smallint	column number
part5	smallint	column number
part6	smallint	column number
part7	smallint	column number
part8	smallint	column number

Index Name	Type	Columns
idxtab	dupls	tabid
idxname	unique	idxname, tabid

The SYSTABAUTH Catalog

Column Name	Type	Explanation
grantor	char(8)	grantor of permission
grantee	char(8)	grantee (receiver) of permission
tabid	integer	table identifier
tabauth	char(7)	authorization type

Index Name	Type	Columns
tabgtor	unique	tabid, grantor, grantee
tabgtee	dupls	tabid, grantee

The SYSCOLAUTH Catalog

Column Name	Type	Explanation
grantor	char(8)	grantor of permission
grantee	char(8)	grantee (receiver) of permission
tabid	integer	table identifier
colno	smallint	column number
colauth	char(2)	authorization type

Index Name	Type	Columns
colgtor	unique	tabid, grantor, grantee, colno
colgtee	dupls	tabid, grantee

The SYSDEPEND Catalog

Column Name	Type	Explanation
btabid	integer	tabid of base table or view
btype	char(1)	base object type (table or view)
dtabid	integer	tabid of dependent table
dtype	char(1)	dependent object type (only view now)

Index Name	Type	Columns
btabid	dups	btabid
dtabid	dups	dtabid

The SYSSYNONYMS Catalog

Column Name	Type	Explanation
owner	char(8)	user name of owner
synonym	char(18)	synonym identifier
created	date	date synonym created
tabid	integer	table identifier

Index Name	Type	Columns
synonym	unique	owner, synonym
syntabid	dupls	tabid

The SYSUSERS Catalog

Column Name	Type	Explanation
username	char(8)	user login identifier
usertype	char(1)	D = DBA, R = RESOURCE, C = CONNECT
priority	smallint	reserved for future use
password	char(8)	reserved for future use

Index Name	Type	Columns
users	unique	username

The SYSVIEWS Catalog

Column Name	Type	Explanation
tabid	integer	table identifier
seqno	smallint	sequence number
text	char(64)	portion of SELECT statement

Index Name	Type	Columns
view	unique	tabid, seqno

Appendix L

Complex Outer Joins

Complex Outer Joins

When more than two tables participate in an outer join, there are many ways to relate them. For three tables, there are five logically distinct joins, of which four involve outer joins. These distinct outer joins are listed later in this appendix using the notation to be used in SELECT statements and a graphic representation that can be generalized for any number of tables. The detailed rules for the graphic representation are located at the end of this section. Briefly, the subservient tables are placed on a lower level than the dominant tables.

The two-table outer join described in Chapter 1 is graphically represented as follows:

```
select a, b
from tab1, outer tab2
where a = b

tab1 ——— ∅          Level 1
          |
        tab2        Level 2
```

In the examples that follow, the tilde (~) symbol represents any Boolean relation between the columns of two tables.

Example 1

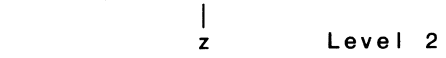
```
from x, y, z
where ...

 x ——— y ——— z      Level 1
```

This is the standard inner join and all three tables are treated on the same level. There are no restrictions on the WHERE clause. It need not exist. Before the WHERE clause is applied, a full Cartesian product is made among all three tables. (**Note**: A Cartesian product of two tables appends every row of the second table to every row of the first. A Cartesian product of three tables is the Cartesian product of the third with the Cartesian product of the first two, and so on.)

Example 2

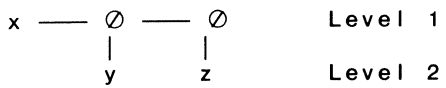
```
from x, y, outer z
where (x or y) ~ z
```



Example 2 illustrates the simplest extension of the outer join from two tables to three tables. **x** and **y** are joined at the same level and are shown connected. **z** is the subservient table and is placed down one level. The WHERE clause must relate a Level 1 table with a Level 2 table. Other relationships are also permitted. Before the WHERE clause is applied, **RDSQL** takes the Cartesian product of **x** and **y** and then takes the outer join of the result with **z**.

Example 3

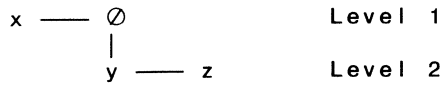
```
from x, outer y, outer z
where x ~ y and x ~ z
```



Although both **y** and **z** are at the same level, they are not connected. Here, a relationship is required between **x** and **y** and also between **x** and **z**. No relationship is permitted between **y** and **z** since they are not connected at the second level. The outer join is first performed between **x** and **y** and then between the resulting table and **z**.

Example 4

```
from x, outer (y, z)
where x ~ (y or z)
```



In Example 4, a relationship is required between **x** and either **y** or **z** or both. In addition, there may be a relationship between **y** and **z** since they are connected at the second level. To perform this join, **RDSQL** starts at the top and works down. For each row of **x**, **RDSQL** attempts to find a row of **y** that satisfies the WHERE clause. If none is found, **RDSQL** substitutes NULL values for the columns of **y** and **z**. If a row of **y** satisfies the WHERE clause, **RDSQL** seeks a row of **z** that satisfies the WHERE clause. It substitutes NULL values for both **y** and **z** if no row in **z** is found.

Example 5

```
from x, outer (y, outer z)
where x ~ y and y ~ z

 x ---- ∅              Level 1
        |
        y ---- ∅       Level 2
               |
               z       Level 3
```

Example 5 shows a three-level outer join. There must always be a relationship between adjacent levels. In this case, **x** must be related to **y**, and **y** must be related to **z**. **RDSQL** performs this join by picking a row from **x** and looking for a pair from **y** and **z** that satisfies (**y**, outer **z**).

If each of the tables consists of a single column with the values in Figure L-1, then the previously described joins result in the subsequent tables.

x.a	y.b	z.c
1	2	3
2	3	4
3	4	5
5		

Figure L-1. Initial Tables

Example 1

```
select *
from x, y, z
where x.a = z.c
```

a	b	c
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-2. Example 1 Results

Example 2

```
select *
from x, y, outer z
where x.a = z.c
```

a	b	c
1	2	-
1	3	-
1	4	-
2	2	-
2	3	-
2	4	-
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-3. Example 2 Results

Example 3

```
select *
from x, outer y, outer z
where x.a = y.b and x.a = z.c
```

a	b	c
1	-	-
2	2	-
3	3	3
5	-	5

Figure L-4. Example 3 Results

Example 4

```
select *
from x, outer (y, z)
where x.a = z.c
```

a	b	c
1	-	-
2	-	-
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-5. Example 4 Results

Example 5

```
select *
from x, outer (y, outer z)
where x.a = y.b and y.b = z.c
```

a	b	c
1	-	-
2	2	-
3	3	3
5	-	-

Figure L-6. Example 5 Results

The following steps are useful when considering how **RDSQL** performs an outer join:

1. Draw the graph corresponding to the FROM clause. Replace each keyword OUTER with the symbol ∅ and put the table names to which the keyword applies in the next lower level.

2. Ensure that a condition exists in the WHERE clause relating a table on each level through each ∅ to a table in the next level below.

3. Form a Cartesian product of all tables connected on the same level, applying those conditions of the WHERE clause that apply only to that level.

4. Starting with the first level, take a row at a time from the resulting table and attempt to satisfy the WHERE clause with the resulting table of the next lower level, replacing the columns of the lower level table with NULL values if the WHERE clause cannot be satisfied.

For example, the following FROM clause results in the accompanying graph:

```
from x, outer (y, outer (z, a, b)), outer c, d

    x ——— ∅ ——————— ∅ ——— d          Level 1
          |         |
          y ——— ∅   c                Level 2
                |
                z ——— a ——— b        Level 3
```

A Cartesian product is made between **x** and **d** (call it **txd**) and between **z**, **a**, and **b** (call it **tzab**), since these sets of tables are connected on the same level. **y** and **c** are both on Level 2, but they are not connected on that level. For each row of **txd**, **RDSQL** attempts to find a row of **y** that satisfies the WHERE clause. If it succeeds, it searchs for a row of **tzab** that satisfies the WHERE clause. In both cases, NULL values are substituted if no satisfactory row is found. **RDSQL** then searches for a row of **c** that satisfies the WHERE clause, substituting NULL values if unsuccessful.

Appendix M

Accessing Programs from the Operating System

Program Command Line Syntax

You can access the modules that comprise INFORMIX-SQL in three ways:

- From the INFORMIX-SQL Main Menu
- Directly from the operating system command line using the module names
- Directly from the operating system command line using a shortened version of the INFORMIX-SQL Main Menu options

The INFORMIX-SQL Main Menu is described in Chapter 1, "The INFORMIX-SQL Main Menu." Using INFORMIX-SQL from the Main Menu is discussed in the *INFORMIX-SQL User Guide.*

DOS users should be aware that accessing programs from the command line requires less memory than accessing programs through the Main Menu or through the **User-menu** option. Because on most DOS systems memory space is limited, you should consider accessing programs from the command line when possible.

If you are accessing a program through either the Main Menu or the **User-menu** option, and you receive an error message indicating that you have run out of space in memory, exit to the command line. This action clears memory. Then enter isql from the command line and you will be able to resume your operations.

How to access the INFORMIX-SQL modules using the operating system command line syntax for each program is described later in this appendix.

The method using the shortened version of the **INFORMIX-SQL** Main Menu options from the command line is described next.

For easy reference, the major **INFORMIX-SQL** Main Menu options follow.

Form	Run, Modify, Generate, New, Compile, Drop
Report	Run, Modify, Generate, New, Compile, Drop
Query-language	New, Run, Modify, User-editor, Output, Choose, Save, Info, Drop
Database	Create, Drop
User-menu	Run, Modify
Table	Create, Alter, Info, Drop

Figure M-1. INFORMIX-SQL Main Menu Options

Accessing Form Menu Options

The operating system command line syntax for accessing the INFORMIX-SQL Form Menu options is

---

isql [**-s**] [*database-name*] **-f**[*options*] [*form-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-f	calls the **Form** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the Form Menu options you select.
form-name	is the name of the form you want to access. Do not include an extension.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following command runs the **customer** form.

```
isql -fr customer
```

The following generates a form based on the **stores** database.

```
isql stores -fg
```

Accessing Report Menu Options

The operating system command line syntax for accessing INFORMIX-SQL Report Menu options is

---

isql [**-s**] [*database-name*] **-r**[*options*] [*report-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-r	calls the **Report** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the Report Menu options you select.
report-name	is the name of the report you want to access. Do not include an extension in the report name.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following command compiles the **clist1** report.

```
isql -rc clist1
```

Accessing Query-Language Menu Options

The operating system command line syntax for accessing INFORMIX-SQL Query-language Menu options is

---

isql [**-s**] [*database-name*] **-q**[*options*] [*command-file-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-q	calls the **Query-language** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the Query-language Menu options you select.
command-file	is the name of the **.sql** file that you want to access. Do not include an extension in the command filename.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following command chooses the **ex1** file and makes the RDSQL statements it contains the current statements.

```
isql -s stores -qc ex1
```

Accessing Database Menu Options

The operating system command line syntax for accessing INFORMIX-SQL Database Menu options is

---

isql [**-s**] [*database-name*] **-d**[*options*] [*database-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-d	calls the **Database Menu** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the Database Menu options you select.
database-name	is the name of the database you want to access.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following statement calls the **Select** option on the DATABASE Menu.

```
isql -s -ds
```

Accessing User-Menu Options

The operating system command line syntax for accessing INFORMIX-SQL User-menu options is

---

isql [**-s**] [*database-name*] **-u**[*options*] [*user-menu-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-u	calls the **User-menu** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the User-menu Menu options you select.
user-menu-name	is the name of the user-menu you want to run.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following command runs the User-menu for the **stores** database.

```
isql stores -ur
```

Accessing Table Menu Options

The operating system command line syntax for accessing INFORMIX-SQL Table Menu options is

---

isql [**-s**] [*database-name*] **-t**[*options*] [*table-name*]

---

isql	is the program call for INFORMIX-SQL.
-s	calls the *silent* option and suppresses all non-essential screen messages.
database-name	is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
-t	calls the **Table** option from the INFORMIX-SQL Main Menu.
options	are the first letters of the Table Menu options you select.
table-name	is the name of the table you want to access.

Notes

1. INFORMIX-SQL returns you to the operating system after you complete the specified operation.

Examples

The following command creates a table in the **stores** database.

```
isql -s stores -tc
```

FORMBUILD

The command syntax for creating a customized screen form directly from the operating system is

---

sformbld [**-s**] [**-v**] { *filename* | **-d** }

---

sformbld is the program call for **FORMBUILD**.

-s calls the *silent* option and suppresses all non-essential screen messages.

-v tells **FORMBUILD** to verify that the fields contained in the screen section of the form specification are consistent with the field widths of the corresponding columns.

FORMBUILD reports any discrepancies in the file **filename.err**.

filename is the name of the form specification file. Do not include the **.per** extension (**filename.per**) on the command line.

-d replaces *filename* and instructs **FORMBUILD** to prompt you for the information required to create and compile a default form specification.

Notes

1. If the compilation is successful, **FORMBUILD** creates a compiled form specification named **filename.frm**. You can use this compiled form specification with **PERFORM** as a screen form. If the compilation is unsuccessful, **FORMBUILD** creates an error file named **filename.err**. You must edit the error file, remove the error messages, and recompile with **FORMBUILD** before you can use the screen form.

2. You can also create a customized screen form directly from the operating system using the shortened version of the **INFORMIX-SQL** Main Menu options. This method is described earlier in this appendix.

PERFORM

The command syntax for running a compiled screen form directly from the operating system is

---

sperform *filename* [...]

---

sperform	is the program call for **PERFORM**.
filename	is the name of the compiled form specification file. Do not include the **.frm** extension (**filename.frm**) on the command line.

Notes

1. The maximum number of *filenames* you can include on the command line is operating system dependent.

2. **PERFORM** displays each form in the order that it appears on the command line.

3. If **PERFORM** cannot display a form, it aborts. When multiple filenames are included on the command line, subsequent filenames are not displayed.

4. You can also run a compiled screen form directly from the operating system using the shortened version of the **INFORMIX-SQL** Main Menu options. This method is described earlier in this appendix.

ACEPREP

The command syntax for compiling a customized report form directly from the operating system is

---

saceprep [**-s**] [**-o** *directory-name*] *filename*

---

saceprep	is the program call for **ACEPREP**.
-s	calls the *silent* option and suppresses all non-essential screen messages.
-o *directory-name*	tells **ACEPREP** to place the output file (either the compiled report specification or the error file) in the indicated directory.
filename	is the name of the report specification file. Do not include the **.ace** extension (**filename.ace**) on the command line.

Notes

1. If the compilation is successful, **ACEPREP** creates a compiled report specification file named **filename.arc**. You can use this compiled report specification with **ACEGO** to produce a report. If the compilation is unsuccessful, **ACEPREP** creates an error file named **filename.err**. You must edit the error file, remove the error messages, and recompile with **ACEPREP** before you can run the report.

2. You can also compile a customized report form directly from the operating system using the shortened version of the **INFORMIX-SQL** Main Menu options. This method is described earlier in this appendix.

ACEGO

The command syntax for running a compiled report directly from the operating system is

---

sacego [**-s**] [**-d** *database-name*] *filename* [...]

---

sacego	is the program call for ACEGO.
-s	calls the *silent* option and suppresses all non-essential screen messages.
-d *database-name*	overrides the database that is named in the report specification and substitutes *database-name.*
filename	is the name of the compiled report specification. Do not include the **.ace** extension (**filename.ace**) on the command line.

Notes

1. The maximum number of *filenames* you can include on the command line is operating system dependent.

2. ACEGO executes each report in the order in which it appears on the command line.

3. If ACEGO cannot execute a report, it aborts. When multiple filenames are included on the command line, subsequent filenames are not executed.

4. You can also run a compiled report directly from the operating system using the shortened version of the INFORMIX-SQL Main Menu options. This method is described earlier in this appendix.

INFORMIX-SQL
Error Messages

This section contains the text of the **INFORMIX-SQL** error messages. It also describes how the system responds to each error and suggests corrective actions.

All **INFORMIX-SQL** errors include an error number. With the error number you can quickly locate the message in this section. Error messages with negative numbers appear in order, starting with −100. Error messages with positive numbers (if any) are placed at the back of the section.

Most **INFORMIX-SQL** error messages appear as a positive number on the screen, even though the originating error code has a minus flag before the number. In most programs, this flag is removed by **INFORMIX-SQL** before the message is displayed. Because the original error code is useful in some situations, this section includes the minus flag before the error numbers. When you wish to locate an error message in this section, you will usually need to look for the negative of the number that is displayed.

-100 **Description of Error**: C-ISAM error: there is already a record with the same value in a unique key.

System Action Taken: The statement was not processed.

Corrective Action: Check that you did not attempt to add a duplicate value to a column with a unique index via *iswrite*, *isrewrite*, *isrewcurr*, or *isaddindex*.

-101 **Description of Error**: C-ISAM error: file is not open.

System Action Taken: The statement was not processed.

Corrective Action: Check that the C-ISAM file has been opened using the *isopen* call, or that you have not tried to write to a C-ISAM file opened for read only.

-102 **Description of Error**: C-ISAM error: illegal argument to C-ISAM function.

System Action Taken: The statement was not processed.

Corrective Action: Check that one of the arguments to the C-ISAM call is not outside of the range of acceptable values for that argument.

-103 **Description of Error**: C-ISAM error: illegal key descriptor (too many parts or too long).

System Action Taken: The statement was not processed.

Corrective Action: Check that one or more of the elements that make up the key description is not outside of the range of acceptable values for that element. (There is a maximum of 8 parts and 120 characters to each key descriptor.)

-104 **Description of Error**: C-ISAM error: too many files open.

System Action Taken: The statement was not processed.

Corrective Action: The maximum number of files that may be open at one time would be exceeded if this request were processed. Reduce the number of open files by breaking your program up into two or more parts. On UNIX systems (and most DOS systems), the maximum number of open files per process is 20. If running under DOS, check that your **CONFIG.SYS** file contains the line

```
FILES=20
```

-105 **Description of Error**: C-ISAM error: bad ISAM file format.

System Action Taken: The statement was not processed.

Corrective Action: The format of the C-ISAM file has been corrupted. Run the **bcheck** utility on the file, and it will try to repair damaged indexes. If **bcheck** cannot repair the file, you will need to reload your data from a backup medium.

-106 **Description of Error**: C-ISAM error: non-exclusive access.

System Action Taken: The statement was not processed.

Corrective Action: You must first open the file with exclusive access when you wish to add or delete an index.

-107 **Description of Error**: C-ISAM error: record is locked.

System Action Taken: The statement was not processed.

Corrective Action: The record or file requested by this call cannot be accessed because it has been locked by another user. Wait a moment and reenter your request.

If you are certain that the table is not in use, you may need to empty the contents of the *tablename.lok* file. (This file contains information about which rows of the table are being used at any one time. It is normally emptied when a user finishes accessing the table. On occasion, the file is not emptied and, as a result, no one will be able to use the table.) You can copy the "file" **/dev/null** into this file to remove all locks on rows in the table. Contact your system administrator about this action.

-108 **Description of Error**: C-ISAM error: key already exists.

System Action Taken: The statement was not processed.

Corrective Action: You have attempted to add an index that previously has been defined. You will need to delete this existing index before defining another.

-109 **Description of Error**: C-ISAM error: the key is the file's primary key.

System Action Taken: The statement was not processed.

Corrective Action: An attempt was made to delete the primary key column. The primary key may not be deleted by the *isdelindex* call.

-110 **Description of Error**: C-ISAM error: end or beginning of the file.

System Action Taken: The statement was not processed.

Corrective Action: The beginning or end of the file was reached.

-111 **Description of Error**: C-ISAM error: no record found.

System Action Taken: The statement was not processed.

Corrective Action: No record could be found that contained the requested value in the specified position. Edit your request and reenter.

-112 **Description of Error**: C-ISAM error: there is no current record.

System Action Taken: The statement was not processed.

Corrective Action: An attempt to access a record in the current list has been made, but there is no current list. You must first perform a query and obtain a current list.

-113 **Description of Error**: C-ISAM error: the file is locked.

System Action Taken: The statement was not processed.

Corrective Action: The table you wish to alter is currently being used by another user in exclusive mode. Wait until the table is no longer being used before entering your request.

If you are certain that the table is not in use, you should run the RDSQL UNLOCK TABLE command to

unlock the table. Also, if your system contains *tablename.lok* files, you may need to empty the contents of the *tablename.lok* file. (This file contains information about which rows of the table are being used at any one time. It is normally emptied when a user finishes accessing the table. On occasion, the file is not emptied and, as a result, no one will be able to use the table.) You can copy the "file" **/dev/null** into this file to remove all locks on rows in the table. Be certain no processes are accessing the locked table before emptying the *tablename.lok* file. Contact your system administrator about this action.

-114 **Description of Error**: C-ISAM error: the filename is too long.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the filename length to 8 or fewer characters (if using DOS) or 10 or fewer characters (if using UNIX).

-116 **Description of Error**: C-ISAM error: cannot allocate memory.

System Action Taken: The statement was not processed.

Corrective Action: Insufficient memory is available to run your request. (INFORMIX-SQL has run out of addressable data space.) You need to reduce the complexity of your statement or form.

-118 **Description of Error**: Cannot read transaction log record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Run the **dblog** utility to determine which record contains the problem.

-119 **Description of Error**: Bad log record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Run the **dblog** utility to determine which record contains the problem.

-120 **Description of Error**: Cannot open log file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the file exists, that you are using the correct pathname, and that you have the appropriate permissions to use the file.

-121 **Description of Error**: Cannot write log file record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have the appropriate file permissions.

-122 **Description of Error**: BEGIN WORK encountered in a database without transactions.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that you have CREATEd or STARTed your database with transactions.

-123 **Description of Error**: No shared memory.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the Database Administrator has set up the shared memory partition.

-124 **Description of Error**: No BEGIN WORK found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must execute BEGIN WORK before COMMIT WORK or ROLLBACK WORK.

-125 **Description of Error**: Cannot use NFS.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not attempt to access remote files on the network using NFS.

-126 **Description of Error**: Audit trail exists.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot specify a new audit trail on a table without dropping the current one.

-200 **Description of Error**: Identifier is too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Identifiers must be 18 characters or less. Select a new identifier of the appropriate length.

-201 **Description of Error**: A syntax error has occurred.

System Action Taken: The statement containing the syntax error was not processed.

Corrective Action: Check that you have not misspelled an RDSQL statement, placed key words out of sequence, or included an INFORMIX-SQL reserved word in your query.

-202 **Description of Error**: An illegal character has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the illegal character (often a non-printable control character) and resubmit the statement.

-203 **Description of Error**: An illegal integer has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Integers must be whole numbers from -2,147,483,647 to 2,147,483,647. Check that you have not included a number with a fractional portion or a number outside of the range of acceptable whole numbers. Check also that you have not inadvertently entered a letter in place of a number (for example, 125p3 instead of 12503).

-204 **Description of Error**: An illegal floating-point number has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not inadvertently entered a letter in place of a number (for example, 125b3 in place of 12503).

-205 **Description of Error**: Cannot use ROWID for views.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Restructure your statement so that the virtual column is not used in defining the view.

-206 **Description of Error**: The specified table name is not in the database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table name in your statement. Check the *systables* system catalog for a list of all database tables.

-208 **Description of Error**: Memory allocation failed during query processing.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the complexity of your query or program.

-209 **Description of Error**: Incompatible database format.

System Action Taken: The statement was not processed.

Corrective Action: You are attempting to use INFORMIX-SQL with a database built for an earlier version of INFORMIX-SQL. You must first run **dbupdate** on your database, which will prepare the database for your current version of INFORMIX-SQL.

-210 **Description of Error**: Pathname too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: INFORMIX-SQL will accept a pathname of up to 70 total characters. Reduce the length of the pathname.

-211 **Description of Error**: Cannot read system catalog *catalog-name*.

System Action Taken: See below for a list of system actions.

Corrective Action: Check the C-ISAM error for information about the source of the problem. Depending upon the content of the statement and the system catalog cited in the error message, the following actions have occurred:

For a CREATE TABLE statement: the **systabauth** catalog was not read, the table was created, but no authorizations are granted to PUBLIC.

For a DROP TABLE statement: if the **systables** catalog was not read, then no action was taken; if the **sysviews** catalog was not read, then the table was dropped but any views built on the table may not have been dropped.

For a DROP VIEW statement: the **sysviews** catalog was not read, and no action was taken.

For a DROP INDEX statement: the **sysindexes** or **systables** catalogs was not read, and the index was not dropped.

For a DROP SYNONYM statement: the **syssynonym** catalog was not read, and the synonym was not dropped.

For a DROP DATABASE statement: the **systables** catalog was not read, and the database was not dropped.

For a START DATABASE statement: the **systables** catalog was not read, and the database was not started.

For a DATABASE statement: the **systables** or **sysusers** catalog was not read, and the database was not selected.

-212 **Description of Error**: Cannot add index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-213 **Description of Error**: Statement interrupted by user.

System Action Taken: The statement was not processed.

Corrective Action: INFORMIX-SQL has received an interrupt signal (probably due to the user pressing the DEL key). Resubmit your statement.

-214 **Description of Error**: Cannot remove file for table *tablename*.

System Action Taken: If this is a DROP DATABASE statement, then some tables may have been dropped from the database. If this is a DROP TABLE statement, then some system entries for the table may have been dropped from the database.

Corrective Action: INFORMIX-SQL cannot remove one or more of the system catalogs that makes up a table. Check the C-ISAM error for information about the source of the problem. Check with the system administrator about remedial actions.

-215 **Description of Error**: Cannot open file for table *tablename*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-216 **Description of Error**: Cannot remove ISAM index on file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-217 **Description of Error**: Column *column-name* not found in any table in the query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Correct the spelling of the column name or that column name exists in database table. Check for the presence of required commas and quotes.

-218 **Description of Error**: Synonym *name* not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the synonym. If needed, check the **syssynonyms** system catalog for a list of available synonyms.

-219 **Description of Error**: Wildcard matching may not be used with non-character types.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Wildcards (*, ?) and characters enclosed in brackets [ ] can be used only with CHAR data types. Check the data type for the offending column.

-220 **Description of Error**: There is no FROM clause in the query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Must include a FROM clause in the query. Check that you do not have an illegal character ($, #, &, etc., or a CONTROL character) in the line prior to the FROM keyword.

-221 **Description of Error**: Cannot build temporary file for new table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: C-ISAM cannot access the **/tmp** directory or the disk may be out of space. Check the C-ISAM error for information about the source of the problem.

-222 **Description of Error**: Cannot write to temporary file for new table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The disk may be out of space. Check the C-ISAM error for information about the source of the problem.

-223 **Description of Error**: Duplicate table name *table-name* in the FROM clause.

System Action Taken: The statement did not run.

Corrective Action: Remove the redundant table name from the statement or use an alias to rename one of the tables.

-224 **Description of Error**: Cannot open log file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-225 **Description of Error**: Cannot create file for system catalog *catalog-name*.

System Action Taken: The CREATE DATABASE statement was not completed. Some system catalogs may have been created.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-226 **Description of Error**: Cannot create index for system catalog *catalog-name*.

System Action Taken: The CREATE DATABASE statement was not completed. Some system catalogs may have been created.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-227 **Description of Error**: Cannot use ORDER BY clause when selecting into temporary table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the ORDER BY clause from your statement. Place an index on the column you wish to order by after creating the temporary table.

-228 **Description of Error**: Cannot have negative characters.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not included a negative CHAR data type (for example, a **–a** or **–p**) in your statement.

-229 **Description of Error**: Could not open or create a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-230 **Description of Error**: Could not read a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-231 **Description of Error**: Cannot perform aggregate function with distinct on expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Select the expression into a temporary table and then perform an aggregate distinct on the temporary table.

-232 **Description of Error**: A SERIAL column *column-name* may not be updated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The values appearing in a SERIAL column are provided by INFORMIX-SQL and may not be updated.

-233 **Description of Error**: Cannot read record that is locked by another user.

System Action Taken: Your request was not processed.

Corrective Action: Another user has locked the record. Wait a moment and reenter your request.

-234 **Description of Error**: Cannot insert into virtual column *column-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The specified column is derived from an expression or an aggregate function. Redefine the view.

-235 **Description of Error**: Character column size too big. The maximum size is 32,767.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Redefine the size of the column.

-236 **Description of Error**: Number of columns in INSERT does not match number of VALUES.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the number of columns in the table or in the column list matches the number of values in the VALUES clause or the SELECT clause.

-237 **Description of Error**: Cannot begin work.

System Action Taken: Your request was not processed.

Corrective Action: Check the C-ISAM error number for information about the source of the problem. Exit the INFORMIX-SQL session. Contact your system administrator or Database Administrator if you need assistance with interpreting the C-ISAM error number.

-238 **Description of Error**: Cannot COMMIT WORK.

System Action Taken: Your request was not processed.

Corrective Action: Your log file may be corrupted. Check the C-ISAM error number for information about the source of the problem. Contact your system administrator or Database Administrator if you need assistance with interpreting the C-ISAM error number.

-239 **Description of Error**: Could not insert new row—duplicate value in a UNIQUE INDEX column.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The row contains a value which already exists in the column (indexed as unique) of an existing row. Enter a new value for the column or remove the unique index on the column.

-240 **Description of Error**: Could not delete a row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-241 **Description of Error**: Cannot ROLLBACK WORK.

System Action Taken: Your request was not processed.

Corrective Action: Check the C-ISAM error number for information about the source of the problem. Exit the INFORMIX-SQL session. Contact your system administrator or Database Administrator if you need assistance with interpreting the C-ISAM error number.

-242 **Description of Error**: Could not open database table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-243 **Description of Error**: Could not position within a table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-244 **Description of Error**: Could not do a physical-order read to fetch next row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-245 **Description of Error**: Could not position within a file via an index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-246 **Description of Error**: Could not do an indexed read to get the next row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-247 **Description of Error**: ROLLFORWARD database failed.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem. Contact your system administrator or Database Administrator if you need assistance with interpreting the meaning of the C-ISAM error.

-249 **Description of Error**: Virtual column must have explicit name.

System Action Taken: The statement containing the error was not processed.

Corrective Action: When selecting into a temporary table or creating a view, each temporary or view column based on an expression must be given a unique name. Check also that distinct names are provided.

-250 **Description of Error**: Cannot read record from file for update.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The record may be locked by another user. Check the C-ISAM error for information about the source of the problem.

-251 **Description of Error**: Column number *number* too big.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of the column noted in your ORDER BY or GROUP BY statement exceeds the total number of columns in the SELECT statement.

-252 **Description of Error**: Cannot get system information for table.

System Action Taken: Some statistics may be updated.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-253 **Description of Error**: Identifier too long—maximum length is 18.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling or length of the table name. in the fetch is equal to the number used when defining the cursor.

-255 **Description of Error**: Not in transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The statement must be executed within a transaction. First start a transaction, then perform the statement.

-256 **Description of Error**: Transaction not available.

System Action Taken: The statement containing the error was not processed.

Corrective Action: INFORMIX-SQL cannot perform a transaction operation (BEGIN WORK, ROLLBACK WORK, COMMIT WORK) on the database because a transaction log was never created for the database. Ask your Database Administrator to create a transaction log for the database.

-258 **Description of Error**: System error - invalid statement ID received by the **sqlexec** process.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Call the Technical Support Department.

-261 **Description of Error**: Could not create file for table *table*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-263 **Description of Error**: Could not lock row for UPDATE.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-264 **Description of Error**: Could not write to a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-265 **Description of Error**: Load or insert cursors must be run within a transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The database is created with transactions. You must first enter a BEGIN WORK statement (starting a transaction), perform the load or insert cursors, and then enter a COMMIT WORK or ROLLBACK WORK statement (closing the transaction).

-268 **Description of Error**: Cannot use SELECT DISTINCT with UNION ALL.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement.

-269 **Description of Error**: Cannot add column *column-name* that does not accept nulls.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement.

-270 **Description of Error**: Could not position within a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-271 **Description of Error**: Could not insert new row into the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-272 **Description of Error**: No SELECT permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to SELECT from the owner of the table.

-273 **Description of Error**: No UPDATE permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to UPDATE from the owner of the table.

-274 **Description of Error**: No DELETE permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to DELETE from the owner of the table.

-275 **Description of Error**: No INSERT permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to INSERT from the owner of the table.

-278 **Description of Error**: Too many ORDER BY columns; maximum is 8.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the number of columns included in the ORDER BY clause to 8 or less.

-279 **Description of Error**: Cannot GRANT or REVOKE database privileges for table or view.

System Action Taken: The statement was not processed.

Corrective Action: Database privileges (Connect, Resource, and DBA) cannot be granted on individual tables. Consult Chapter 2 for appropriate **RDSQL** table privilege statements.

-280 **Description of Error**: Total size of ORDER BY columns exceeds 120 bytes.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the number of columns included in the ORDER BY clause so that the total number of characters is less than or equal to 120 (perhaps delete a CHAR column of 30 or more characters).

-281 **Description of Error**: Could not add index to a temporary table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-282 **Description of Error**: Found a quote for which there is no matching quote.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that all quoted strings are properly terminated with a quote.

-283 **Description of Error**: Found a non-terminated comment ("{" with no matching "}").

System Action Taken: The statement was not processed.

Corrective Action: Check that all comments are properly enclosed in braces. (Comments cannot be nested.)

-284 **Description of Error**: A subquery has returned not exactly one value.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check data for the subquery. Restructure the subquery by adding more components in the WHERE clause so that only one value is returned.

-286 **Description of Error**: An expression may not include ANY or ALL.

System Action Taken: The statement containing the error was not processed.

Corrective Action: ANY and ALL can only be used in association with a subquery.

-287 **Description of Error**: Cannot add SERIAL column *column-name* to the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A SERIAL column does not accept NULL values. Add the column to the table as type INTEGER, UPDATE it so that there are no NULLS, and then MODIFY it to type SERIAL.

-288 **Description of Error**: Could not lock table—it is already in use or cannot be found.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-289 **Description of Error**: Could not lock table *table-name*—table locked by another user.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-290 **Description of Error**: Cursor not declared with FOR UPDATE clause.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have attempted to delete or update WHERE CURRENT OF without declaring the cursor for update. DECLARE your cursor FOR UPDATE.

-291 **Description of Error**: Table *table-name* is already locked.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first unlock the table before executing your request.

-292 **Description of Error**: An implied INSERT column *column-name* does not accept NULLs.

System Action Taken: The statement containing the error was not processed.

Corrective Action: RDSQL will not allow you to insert a NULL value in a column that does not allow NULLS. Check to see if a non-NULL column is included in the column list in the INSERT statement.

-293 **Description of Error**: IS [NOT] NULL predicate may be used only with simple columns.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Restructure your query.

-294 **Description of Error**: The column *column-name* must be in the GROUP BY list.

System Action Taken: The statement was not processed.

Corrective Action: All non-aggregate columns in the SELECT list must be included in the GROUP BY list. Restructure your statement to include all columns that are not aggregate functions.

-295 **Description of Error**: The GROUP BY column number *number* is too large.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of the column noted in your ORDER BY or GROUP BY statement exceeds the total number of columns in the SELECT statement.

-297 **Description of Error**: The SELECT list may not contain a subquery.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the subquery from the SELECT list in the statement.

-298 **Description of Error**: COUNT(DISTINCT ...) may be used only with a simple column.

System Action Taken: The statement was not processed.

Corrective Action: You cannot include expressions with the COUNT(DISTINCT ...) function. Restructure the query.

-299 **Description of Error**: A query may not contain more than one DISTINCT.

System Action Taken: The statement was not processed.

Corrective Action: Restructure your query to include only one DISTINCT.

-300 **Description of Error**: There are too many GROUP BY columns (maximum is 8).

System Action Taken: The request was not completed.

Corrective Action: Reduce the number of columns in the statement to 8 or less.

-301 **Description of Error**: The total size of the GROUP BY columns exceeds 120 characters.

System Action Taken: The request was not completed.

Corrective Action: The total number of characters in all columns listed in the GROUP BY list exceeds 120 characters. Reduce the column list (perhaps exclude any columns with character data types greater than 30 characters.)

-302 **Description of Error**: No GRANT option.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You do not have permission to grant access privileges to the table. Only the table owner or a user with GRANT permission can do this.

-303 **Description of Error**: Expression mixes columns with aggregates.

System Action Taken: The statement was not processed.

Corrective Action: Restructure your query so that columns and aggregates are not included in the same expression.

-304 **Description of Error**: HAVING can only have expressions with aggregates.

System Action Taken: The statement was not processed.

Corrective Action: The HAVING clause can only take on aggregates. Restructure your query.

-305 **Description of Error**: Subscripted column *column-name* is not of type CHAR.

System Action Taken: The request was not completed.

Corrective Action: Remove the subscript delimiter from the non-character column name in the request.

-306 **Description of Error**: Subscript out of range.

System Action Taken: The request was not completed.

Corrective Action: The range of the subscript delimiter exceeds the range of the column data type. Check the size of the data type and reduce the subscript range.

-307 **Description of Error**: Illegal subscript definition.

System Action Taken: The statement did not run.

Corrective Action: Check that you have not reversed the order of the subscript delimiters ([3,8] is a valid subscript; [8,3] is invalid) or included a negative number as a subscript delimiter.

-308 **Description of Error**: Column type must be the same for each UNION statement.

System Action Taken: The statement was not processed.

Corrective Action: Check that each column in the UNION statement are of the same data type.

-309 **Description of Error**: ORDER BY column *column-name* must be in SELECT list.

System Action Taken: The statement did not run.

Corrective Action: Check that columns included in the ORDER BY clause appear in the SELECT list.

-310 **Description of Error**: Table *table-name* already exists in database.

System Action Taken: The request was not completed.

Corrective Action: Select an alternate name for the table.

-311 **Description of Error**: Cannot open system catalog *catalog-name*.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-312 **Description of Error**: Cannot update system catalog *catalog-name*.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-313 **Description of Error**: Not owner of table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only the owner of the table (or the Database Administrator) can remove the table.

-314 **Description of Error**: Table *table-name* currently in use.

System Action Taken: The request was not completed.

Corrective Action: The table you wish to drop is currently being used by another user. Wait until the table is no longer being used before executing your request.

-315 **Description of Error**: No CREATE INDEX permission.

System Action Taken: The request was not completed.

Corrective Action: Permission not granted for you to create an index on the table.

-316 **Description of Error**: Index *index-name* already exists in database.

System Action Taken: The request was not completed.

Corrective Action: An index currently exists for the table. You must drop the existing index before creating a new one.

-317 **Description of Error**: Must have the same number of selected columns in each UNION statement.

System Action Taken: The request was not completed.

Corrective Action: Check the number of columns selected in each SELECT statement.

-318 **Description of Error**: File with the same name as specified log file already exists.

System Action Taken: The request was not completed.

Corrective Action: Select a different name for the log file.

-319 **Description of Error**: Index does not exist in database file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the index name or check the sysindexes system catalog for the correct index name.

-320 **Description of Error**: Not owner of index *index-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only the owner of the index (or the Database Administrator) can remove the index.

-321 **Description of Error**: Cannot group by aggregate column *column-name.*

System Action Taken: The statement was not processed.

Corrective Action: Check the column number used in the GROUP BY clause.

-322 **Description of Error**: Cannot alter view *view-name.*

System Action Taken: The request was not completed.

Corrective Action: Views cannot be altered. You must drop and then recreate the view.

-323 **Description of Error**: Cannot grant permission on temporary table.

System Action Taken: The request was not completed.

Corrective Action: Permissions can only be granted on permanent tables.

-324 **Description of Error**: Ambiguous column *column-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: A column name exists in more than one table cited in your query. Prepend each column name with the appropriate table name.

-325 **Description of Error**: Log file must be given a full pathname starting with a "/".

System Action Taken: The request was not completed.

Corrective Action: Provide the full pathname for the log file.

-327 **Description of Error**: Cannot unload table *table-name* within a transaction.

System Action Taken: The statement was not processed.

Corrective Action: The database has transactions. Issue a BEGIN WORK statement before unloading the table.

-328 **Description of Error**: Column *column-name* already exists in table.

System Action Taken: The request was not completed.

Corrective Action: Select a new column name for the column.

-329 **Description of Error**: Database not found or no system permission.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the database name. Check that the database name exists in your current directory or a directory included in your DBPATH environment variable. Check the C-ISAM error for information about the source of the problem.

-330 **Description of Error**: Cannot create database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not entered the name of an existing database. Select an alternate name for the database. Check the C-ISAM error for information about the source of the problem.

-331 **Description of Error**: Cannot drop database directory.

System Action Taken: All database files in the database directory are deleted, but the directory remains.

Corrective Action: Remove any non-database files present in the database directory, then remove the directory. Check the C-ISAM error for information about the source of the problem.

-332 **Description of Error**: Cannot access audit trail name information.

System Action Taken: The request was not completed.

Corrective Action: Re-enter your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-333 **Description of Error**: The audit trail file already exists with a different name.

System Action Taken: The request was not completed.

Corrective Action: You must first drop the existing audit trail file (issue a DROP AUDIT FOR statement) before creating a new audit trail.

-334 **Description of Error**: Cannot create audit trail.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must indicate the full pathname of the file receiving the audit trail. Check that you have permission to write to a file in the selected directory. Contact your system administrator if you need help with this action.

-335 **Description of Error**: There is no audit trail for the specified table.

System Action Taken: The request was not completed.

Corrective Action: INFORMIX-SQL is unable to recover the table as no audit trail was created.

-336 **Description of Error**: Cannot create or drop audit on temporary table *table-name.*

System Action Taken: The request was not processed.

Corrective Action: You cannot place an audit trail on a temporary table.

-337 **Description of Error**: Cannot create view on temporary table *table-name.*

System Action Taken: The request was not processed.

Corrective Action: You cannot create a view on a temporary table.

-338 **Description of Error**: Cannot drop audit trail.

System Action Taken: The audit trail was not dropped (possible operating system error.)

Corrective Action: Re-enter your request. If the problem reoccurs, check the C-ISAM error message for information about the source of the problem. Contact your system administrator if you require assistance with this action.

-339 **Description of Error**: The audit trail filename must start with '/'.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Edit your statement to include the full pathname of the audit trail file.

-340 **Description of Error**: Cannot open audit trail file.

System Action Taken: The request was not completed (operating system error).

Corrective Action: Check that you have operating system read permission to the file. Contact your system administrator if you need help with this action.

-341 **Description of Error**: Could not read a row from audit trail file.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Reenter your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-343 **Description of Error**: Row from audit trail was added to a different position than expected.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Reenter your request. If you receive the error again, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-344 **Description of Error**: Cannot delete row—row in table does not match row in audit trail.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Reenter your request. If you receive the error again, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-345 **Description of Error**: Cannot update row—row in table does not match row in audit trail.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Reenter your request. If you receive the error again, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-346 **Description of Error**: Could not update a row in the table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-347 **Description of Error**: Could not open table for exclusive access.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-348 **Description of Error**: Could not read a row from the table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-349 **Description of Error**: Database not selected yet.

System Action Taken: The statement containing the error was not processed.

Corrective Action: First select your database, then reenter your command statement.

-350 **Description of Error**: Index already exists on column.

System Action Taken: The request was not completed.

Corrective Action: Adding an index on the column is not necessary, as the column is already indexed.

-351 **Description of Error**: Database contains tables owned by other users.

System Action Taken: The statement was not processed.

Corrective Action: You can only drop a database if you own all tables in the database or have Database Administrator status. Contact your Database Administrator if you need help with this action.

-352 **Description of Error**: Column not found.

System Action Taken: The request was not completed.

Corrective Action: Check the spelling of the column name.

-353 **Description of Error**: No table or view specified when granting or revoking table privileges.

System Action Taken: The request was not completed.

Corrective Action: You must include the name of the table or view on which a privilege is granted or revoked in your **RDSQL** statement.

-354 **Description of Error**: Incorrect database or cursor name format.

System Action Taken: The statement was not processed.

Corrective Action: A database name must be ten characters or less on UNIX systems or eight characters or less on DOS systems. A cursor name must be 18 characters or less on UNIX and DOS systems. A database or cursor name must begin with a letter, and contain letters, numbers, or underscores. Check that you have not included an illegal character in the name.

-355 **Description of Error**: Cannot rename file for table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-356 **Description of Error**: Table "tablename" specified in both main query and subquery.

System Action Taken: The statement was not processed.

Corrective Action: The statement is ambiguous because a column cannot be identified uniquely. Use an alias to rename the offending table.

-357 **Description of Error**: Dependent table for view "viewname" has been altered.

System Action Taken: The statement was not processed.

Corrective Action: A table upon which the view is constructed has been modified (for example, a column has been dropped, a datatype has been modified, or a column has been added to the middle of the table). Drop the view and create a new view.

-358 **Description of Error**: Must close database before CREATE, START, or ROLLFORWARD database.

System Action Taken: The statement was not processed.

Corrective Action: Execute a CLOSE DATABASE statement, then attempt the action.

-359 **Description of Error**: Cannot drop current database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: First execute a CLOSE DATABASE statement before executing a DROP DATABASE statement.

-360 **Description of Error**: Cannot modify table or view used in subquery.

System Action Taken: The statement was not processed.

Corrective Action: If allowed, your statement could reduce to a looping program. Edit your statement.

-361 **Description of Error**: Column size too large.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the size of the column. You cannot have a CHAR column larger than 32,767 characters.

-362 **Description of Error**: Can have only one column of SERIAL type.

System Action Taken: The action was not completed.

Corrective Action: You may not have more than one column of SERIAL type in a table. Select an alternate data type for the column.

-366 **Description of Error**: The scale exceeds the maximum precision specified.

System Action Taken: The request was not completed.

Corrective Action: The problem is located in a DECIMAL or MONEY column: the scale (number of digits to the right of the decimal point) exceeds the precision (total number of digits).

```
DECIMAL(m,n)  where n>m
MONEY(m,n)    where n>m
MONEY(m)      where m<2
```

-367 **Description of Error**: Sums and averages cannot be computed for character columns.

System Action Taken: The request was not completed.

Corrective Action: Check that you have not included a column of character data type in the aggregate function statement.

-368 **Description of Error**: Incompatible sqlexec module.

System Action Taken: The request was not completed.

Corrective Action: Check that the correct version of sqlexec has been installed. Contact your Database Administrator if you need help with this action.

-369 **Description of Error**: Invalid serial number. Please consult your installation instructions.

System Action Taken: The request was not completed.

Corrective Action: Check that the correct version of sqlexec has been installed. Contact your Database Administrator if you need help with this action.

-370 **Description of Error**: Cannot drop last column.

System Action Taken: The request was not completed.

Corrective Action: Only one column remains in the table. Use the DROP TABLE statement to remove the table.

-371 **Description of Error**: Cannot create unique index on column with duplicate data.

System Action Taken: The request was not completed.

Corrective Action: The column contains duplicate data.

-372 **Description of Error**: Cannot alter table with audit trail on.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first drop the audit trail before making any changes to the table. After making the changes you may want to reestablish an audit trail.

-373 **Description of Error**: DBPATH too long.

System Action Taken: The request was not completed.

Corrective Action: Reduce the length of your DBPATH environment variable.

-374 **Description of Error**: Can only use column number in ORDER BY clause with UNION.

System Action Taken: The request was not completed.

Corrective Action: Restructure the query, using ordinal numbers for the ORDER BY columns.

-375 **Description of Error**: Cannot create log file for transaction.

System Action Taken: The request was not completed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-376 **Description of Error**: Log file already exists.

System Action Taken: The request was not completed.

Corrective Action: Select an alternate name for the log file.

-377 **Description of Error**: Must terminate transaction before closing database.

System Action Taken: The request was not completed.

Corrective Action: Issue a COMMIT WORK or ROLLBACK statement before closing the database.

-378 **Description of Error**: Record currently locked by another user.

System Action Taken: The request was not completed.

Corrective Action: Wait until the record is unlocked before submitting the statement. Check the C-ISAM error for information about the source of the problem.

-379 **Description of Error**: Cannot revoke privilege on columns.

System Action Taken: The request was not completed.

Corrective Action: First revoke UPDATE or SELECT privilege on the table, then grant the revoked privileges.

-380 **Description of Error**: Cannot erase log file.

System Action Taken: The request was not completed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-381 **Description of Error**: Cannot grant to someone who has granted you the same privilege before.

System Action Taken: The request was not completed.

Corrective Action: The name of the individual who granted you permission to use the table must be removed from your user list.

-382 **Description of Error**: Same number of columns must be specified for view and select clause.

System Action Taken: The request was not completed.

Corrective Action: Check the number of columns in the view definition and the selected columns.

-383 **Description of Error**: View column for aggregate or expression must be explicitly named.

System Action Taken: The request was not completed.

Corrective Action: Provide a name for all virtual columns.

-384 **Description of Error**: Cannot modify non simple view.

System Action Taken: The request was not completed.

Corrective Action: Can only modify views built on a single table.

-385 **Description of Error**: Data value out of range.

System Action Taken: The request was not completed.

Corrective Action: Check the view definition for valid data ranges.

-386 **Description of Error**: Column contains null values.

System Action Taken: The request was not completed.

Corrective Action: The table contains NULL values in a column being altered to disallow NULLS. Remove NULL values from the column.

-387 **Description of Error**: No connect permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request connect permission.

-388 **Description of Error**: No resource permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request resource permission.

-389 **Description of Error**: No DBA permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request DBA permission.

-390 **Description of Error**: Synonym already used as table name or synonym.

System Action Taken: The request was not completed.

Corrective Action: Select a different synonym. Check the syssynonym system catalog for a list of existing synonyms.

-391 **Description of Error**: Cannot insert a NULL into column *column-name*.

System Action Taken: The request was not completed.

Corrective Action: Check that a column that does not allow NULL values is omitted from the insert column list.

-392 **Description of Error**: System error—unexpected null pointer encountered.

System Action Taken: The request was not completed.

Corrective Action: Contact the Technical Support Department at RDS.

-393 **Description of Error**: A condition in the where clause results in a two-sided outer join.

System Action Taken: The request was not completed.

Corrective Action: A two-sided outer join is not allowed. Restructure your query.

-394 **Description of Error**: View *view-name* not found.

System Action Taken: The request was not completed.

Corrective Action: Check the spelling of the view name. Check the sysviews system catalog for a list of existing views.

-395 **Description of Error**: The where clause contains an outer cartesian product.

System Action Taken: The request was not completed.

Corrective Action: Check the syntax of the statement.

-396 **Description of Error**: Illegal join between a nested outer table and a preserved table.

System Action Taken: The request was not completed.

Corrective Action: Check the syntax of the statement.

-397 **Description of Error**: System catalog corrupted.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator for help with this error.

-399 **Description of Error**: Cannot access log file.

System Action Taken: The request was not completed.

Corrective Action: You may not edit the log file. (one or more have a NULL value). is aligned with the proper address boundary for variables of that type.

-406 **Description of Error**: Memory allocation failed.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, reenter the program you were using, and resubmit your program.

-407 **Description of Error**: Error number zero received from the **sqlexec** process.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, reenter the program you were using, and resubmit your program.

-408 **Description of Error**: Invalid message type received from the **sqlexec** process.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, reenter the program you were using, and resubmit your program.

-409 **Description of Error**: **sqlexec** was not found or was not executable by the current user.

System Action Taken: Your statement did not run.

Corrective Action: Check that your INFORMIXDIR environment variable is properly set. Contact your system administrator if you need help with this action.

-413 **Description of Error**: Insert attempted on unopened cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Open the cursor before executing a PUT statement.

-414 **Description of Error**: Insert attempted on NULL cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you have declared the cursor correctly.

-415 **Description of Error**: Data conversion error discovered during PUT operation.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The host variable is incompatible with the data type of a column in the database. Choose an appropriate host variable or restrict the size of the data in the host variable.

-416 **Description of Error**: USING option with open statement is invalid for insert cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You should use the USING option with the PUT statement.

-417 **Description of Error**: FLUSH can only be used on an insert cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that you are using the correct cursor.

-418 **Description of Error**: NULL SQLDA descriptor or host variable list encountered.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that you are using the correct host variable and descriptor.

-419 **Description of Error**: SQLDATA pointer in SQLDA or host variable is null.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the host variable list to make sure it is valid. Make sure that memory is properly assigned for the SQLDA structure.

-500 **Description of Error**: Clustered index already exists in the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not create a clustered index where one already exists. Alter the existing cluster index to NOT CLUSTER before creating a new clustered index.

-501 **Description of Error**: Index is already not clustered.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You do not need to alter index to NOT CLUSTER.

-502 **Description of Error**: Cannot cluster index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error code for more information about the problem.

-503 **Description of Error**: Too many tables locked.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Either unlock one or more tables, or open the database in exclusive mode.

-504 **Description of Error**: You cannot lock a view.

System Action Taken: Your statement was not executed.

Corrective Action: Edit your program or statements and remove the request to lock the view.

-505 **Description of Error**: Number of columns in UPDATE does not match number of VALUES.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your RDSQL statement.

-506 **Description of Error**: Do not have permission to update all columns

System Action Taken: The statement containing the error was not processed.

Corrective Action: Obtain permission from the owner of the table.

-507 **Description of Error**: Cursor not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: This cursor has not been defined. Check the spelling of the cursor.

-508 **Description of Error**: You cannot rename a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Modify your program, or create another table and insert to fill the table.

-509 **Description of Error**: You can not rename a column in a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Use an alias to define the column name as part of the select statement.

-802 **Description of Error**: Cannot open file for run (operating system error).

System Action Taken: Your statement was not executed.

Corrective Action: Check that the file exists (if it is not found in your current directory you will need to include a full pathname). Check that you have operating system read permission to access the file. Contact your system administrator if you need help with this action.

-804 **Description of Error**: Comment has no end.

System Action Taken: Your statement containing the error was not processed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Edit your statement to include a matching right brace } to the left brace { denoting the start of your comment. (Note: comments may not be nested.)

-805 **Description of Error**: Cannot open file for load.

System Action Taken: Your statement did not run (operating system error).

Corrective Action: Check that the file exists (if it is not found in your current directory you will need to include a full pathname). Check that you have operating system read permission to read the file. Contact your system administrator if you need help with this action.

-806 **Description of Error**: Cannot open file for unload.

System Action Taken: Your statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission in the designated directory. Contact your System Administrator if you need help with this action.

-807 **Description of Error**: Cannot open file for output.

System Action Taken: Your statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission in the designated directory. Contact your System Administrator if you need help with this action.

-808 **Description of Error**: Cannot open file for choose.

System Action Taken: Your statement did not run (operating system error).

Corrective Action: Check that you have operating system read permission to access the file. Contact your System Administrator if you need help with this action.

-809 **Description of Error**: RDSQL syntax error has occurred.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not misspelled an RDSQL statement, placed key words out of sequence, or included an INFORMIX-SQL reserved word in your query.

-810 **Description of Error**: Cannot open file for save.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the designated directory. Contact your System Administrator if you need help with this action.

-811 **Description of Error**: Cannot open printer for output.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have correctly set your DBPRINT environment variable. Contact your System Administrator if you need help with this action.

-812 **Description of Error**: Cannot open pipe for output.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check the spelling of the name of the program receiving the output. Check that the program is available on your system. Check that the program exists in a directory accessed in your PATH environment variable. Contact your System Administrator if you need help with this action.

-813 **Description of Error**: Cannot write to pipe for output (no reading process).

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check the spelling of the name of the program receiving the output. Check that the program is available on your system. Check that the program exists in a directory accessed in your PATH environment variable. Contact your System Administrator if you need help with this action.

-816 **Description of Error**: Cannot write file (check file permissions).

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system "write" permission to access the file. Contact your System Administrator if you need help with this action.

-817 **Description of Error**: Cannot read file (check file permissions).

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system "read" permission in the designated directory. Contact your System Administrator if you need help with this action.

-818 **Description of Error**: Specified user menu not found.

System Action Taken: The statement did not run.

Corrective Action: Check the spelling of the specified user menu name or that the menu exists in your current database.

-819 **Description of Error**: There are no menu items in the menu.

System Action Taken: No menu items are available to execute.

Corrective Action: The current menu contains no items. Modify the user-menu or select another menu.

-820 **Description of Error**: No more data to display.

System Action Taken: INFORMIX-SQL has displayed all requested record(s).

Corrective Action: Re-enter your request if you would like to view the record(s) again.

-821 **Description of Error**: Cannot open file for default report.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the designated directory. Contact your System Administrator if you need help with this action.

-822 **Description of Error**: Statements are already saved.

System Action Taken: The request was not completed.

Corrective Action: The current RDSQL statement has already been saved under the name indicated at the top of the screen.

-823 **Description of Error**: There are no statements to run.

System Action Taken: No statement was run.

Corrective Action: No current statement exists for RDSQL to run. Enter a new statement or choose a statement with the RDSQL Menu **Choose** option.

-824 **Description of Error**: Missing values clause on insert statement.

System Action Taken: The statement did not run.

Corrective Action: The INSERT INTO statement requires a VALUES clause. Check that you have included a VALUES clause (with a values list) in your statement.

-825 **Description of Error**: Program not found.

System Action Taken: The statement did not run (operating system error).

Corrective Action: RDSQL could not locate a necessary program. Check that your INFORMIXDIR environment variable is properly set. Contact your System Administrator if you need help with this action.

-826 **Description of Error**: Fork system call failed.

System Action Taken: The statement did not run (operating system error).

Corrective Action: RDSQL was unable to fork a necessary process. Pause a moment and reenter your request. If you receive this error message again, contact your System Administrator.

-827 **Description of Error**: Database not found.

System Action Taken: The statement was not processed.

Corrective Action: Check the spelling of the database name. Check that the database name exists in your current directory or a directory specified in your DBPATH environment variable.

-828 **Description of Error**: Command file not found.

System Action Taken: The statement was not processed.

Corrective Action: Check the spelling of the command filename. Check that the command filename exists in your current directory or a directory specified in your DBPATH environment variable.

-829 **Description of Error**: Form not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the form name. Make sure the form exists in your current directory or a directory specified in your DBPATH environment variable.

-830 **Description of Error**: Report not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the report name. Make sure the report exists in your current directory or a directory specified in your DBPATH environment variable.

-831 **Description of Error**: Error(s) found in Report specifications.

System Action Taken: The report compile was not completed.

Corrective Action: Edit the report specification file following the prompts provided by **ACEPREP**. Identified error messages may be located in this section.

-832 **Description of Error**: Error(s) found in form specifications.

System Action Taken: The form compile was not completed.

Corrective Action: Edit the form specification file following the prompts provided by **FORMBUILD**. Identified error messages may be located in this section.

-833 **Description of Error**: saceprep could not compile Report.

System Action Taken: The report compile was not completed.

Corrective Action: Attempt a second compile of the report, or run **ACEPREP** outside of **INFORMIX-SQL**.

-834 **Description of Error**: **sformbld** could not compile the form.

System Action Taken: The form compile was not completed.

Corrective Action: Attempt a second compile of the form, or run **FORMBUILD** outside of **INFORMIX-SQL**.

-835 **Description of Error**: Current clause is invalid in interactive mode.

System Action Taken: The statement did not run.

Corrective Action: The WHERE CURRENT OF clause cannot be run interactively in **INFORMIX-SQL**. Remove the WHERE CURRENT OF clause from the statement.

-836 **Description of Error**: Insert statement has no values clause.

System Action Taken: The statement did not run.

Corrective Action: The INSERT INTO statement requires a VALUES clause. Check that you have included a VALUES clause (with a values list) in your statement.

-837 **Description of Error**: There is not enough memory available.

System Action Taken: The statement did not run.

Corrective Action: There is insufficient data space in memory to run your request. Save your statement, exit from **INFORMIX-SQL**, and then reenter **INFORMIX-SQL** and resubmit the statement. If this does not work you may need to reduce the complexity of your statement.

-838 **Description of Error**: A line in the load file is too long.

System Action Taken: The statement did not run.

Corrective Action: Insufficient data space is available in memory to run your request. Save your statement, exit from **INFORMIX-SQL**, and then reenter **INFORMIX-SQL** and resubmit the statement.

-839 **Description of Error**: Table not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table name. Make sure the *table-name.dat*, *table-name.idx*, and *table-name.lok* files are located in the *database-name.dbs* directory.

-840 **Description of Error**: Name is too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Database and report names must be eight (in DOS) or ten (in UNIX) characters or less. Select a new name of the appropriate length.

-841 **Description of Error**: Name must start with a letter and contain letters, digits, or underscores.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Select a name beginning with a letter and containing only letters, digits, and underscores.

-842 **Description of Error**: Cannot read temp file.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system "read" permission to access the file. Contact your System Administrator if you need help with this action.

-843 **Description of Error**: Cannot write temp file.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system "write" permission to create a file in the current directory, the **/tmp** directory, or the directory indicated in your DBTEMP environment variable. Contact your System Administrator if you need help with this action.

-845 **Description of Error**: There are no user-menus in the database.

System Action Taken: The request was not completed and you are returned to the **INFORMIX-SQL** main menu.

Corrective Action: No user-menus have been created for the current database. Select an alternate database. If a user-menu does exist in the current directory, check that the top-level menu is named "main."

-846 **Description of Error**: Number of values in load file is not equal to number of columns.

System Action Taken: The statement did not run.

Corrective Action: Check that the number of values in the load file is equal to the number of columns present in the table you are loading into.

-1200 **Description of Error**: Number is too large for a DECIMAL data type.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The range of numbers allowed as a decimal data type has been exceeded. Allowable decimal numbers range from $-.1 \times 10^{-128}$ to $.1 \times 10^{126}$ (with 32 significant digits). Check the size of the number.

-1201 **Description of Error**: Number is too small for a DECIMAL data type.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The range of numbers allowed as a decimal data type has been exceeded. Allowable decimal numbers range from $-.1 \times 10^{-128}$ to $.1 \times 10^{126}$ (with 32 significant digits). Check the size of the number.

-1202 **Description of Error**: An attempt was made to divide by zero.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you are not attempting to divide a numerical column type by a character column type (for example, `16/Jones`) or that the value of the divisor does not equal zero.

-1203 **Description of Error**: Values used in a MATCH must both be type CHARACTER.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the values included in your MATCH condition are both CHAR types. Use an alternate comparison condition for non-CHAR types.

-1204 **Description of Error**: Invalid year in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Acceptable years are 0001 to 9999. If two digits are used, RDSQL assumes the year is 19xx. Check the value entered in the date field.

-1205 **Description of Error**: Invalid month in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Months must be represented as the number of the month (1 through 12). Check the value entered in the date field.

-1206 **Description of Error**: Invalid day in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Acceptable days are 01 through 31. Check the value entered in the date field.

-1208 **Description of Error**: Returned by **rvaldata()**: The intended destination type (character) is not the same type as the source. There is no conversion from non-character values to character values.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the data types of both columns are the same.

-1209 **Description of Error**: Without any delimiters, this date must contain exactly 6 or 8 digits.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must enter either 6 or 8 digits when specifying a data value to represent the date.

-1210 **Description of Error**: Date could not be converted to month/day/year format.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Dates must be presented as month, day, and year (August 4, 1985 is allowed; 4 August, 1985 is not). Check the sequence of characters entered in the date field.

-1211 **Description of Error**: Out of memory.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have exceeded the data space capacity on your machine. Reduce the complexity of your form.

-1212 **Description of Error**: Date conversion format string must contain a month, day, and year component.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The FORMAT string used to format a DATE field must contain month, day, and year components. One of these is missing.

-1213 **Description of Error**: Character to numeric conversion error.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the values in the character string contain only ASCII characters representing numeric data types. (A table of ASCII codes is included as an appendix to this manual.)

-1214 **Description of Error**: Value too large to fit in a SMALLINT.

System Action Taken: The statement was not processed.

Corrective Action: Acceptable SMALLINT values are whole numbers between -32,767 and 32,767. If a larger number is required, you need to use the **RDSQL** ALTER TABLE statement to modify the column to INTEGER type.

-1215 **Description of Error**: Value too large to fit in an INTEGER.

System Action Taken: The statement was not processed.

Corrective Action: Acceptable INTEGER values are whole numbers between -2,147,483,647 and 2,147,483,647. If a larger number is required, you need to use the RDSQL ALTER TABLE statement to modify the column to DECIMAL type.

-1216 **Description of Error**: Illegal exponent.

System Action Taken: The statement was not processed.

Corrective Action: Check that the exponent is an integer with a value not exceeding 32,767.

-1217 **Description of Error**: The format string is too large.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the size of the FORMAT string (used to format a DATE field) in the form specification.

-1218 **Description of Error**: String to date conversion error.

System Action Taken: The statement was not processed.

Corrective Action: Check the format of the DATE data type in the DBDATE environment variable. The format is illegal.

-1226 **Description of Error**: Decimal or money value exceeds maximum precision.

System Action Taken: The statement was not processed.

Corrective Action: Increase the precision of the DECIMAL or MONEY field.

-2014 **Description of Error**: There was an incorrect number of arguments on the operating system command line. At least one argument is expected.

System Action Taken: The operating system command was not executed.

Corrective Action: **FORMBUILD** requires that you include a filename on the command line (unless you use the **-d** option to **FORMBUILD**). Carefully reenter the command, including the filename as an argument, or use the **-d** option to **FORMBUILD**.

-2015 **Description of Error**: An open comment symbol, {, was found inside an already open comment on line *lineno*, character *charposition*. This could be due to a failure to close the previously opened comment, which was begun on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Insert a close comment symbol where appropriate. (Note: comments may not be nested.)

-2016 **Description of Error**: A comment has been opened, but not closed. The last comment begun was opened on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Insert a close comment symbol where appropriate.

-2018 **Description of Error**: A grammatical error has been found on line *lineno*, character *charposition*. The construct is not understandable in its context.

System Action Taken: The compile was not completed.

Corrective Action: Check the grammatical content of the statement (placement of commas, braces, and so on).

-2019 **Description of Error**: This integer exceeds the maximum size allowed.

System Action Taken: The compile was not completed.

Corrective Action: Allowable INTEGER values are whole numbers that range from -2,147,483,647 to 2,147,483,647. Check the value of the number (number of digits and location of the decimal point). If a larger number is required, you will need to use the **RDSQL** ALTER TABLE statement to modify the column to DECIMAL type.

-2020 **Description of Error**: The table *tablename* could not be opened. The operating system was asked to open it for writing.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the designated directory. Contact your System Administrator if you need help with this action.

-2022 **Description of Error**: This identifier exceeds the maximum length for identifiers, which is 50.

System Action Taken: The compile was not completed.

Corrective Action: Check that all field names, field labels, and identifiers are less than or equal to 50 characters in length.

-2023 **Description of Error**: This quoted string exceeds the maximum length for quoted strings, which is 80.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of characters in the quoted string to 80 or less.

-2025 **Description of Error**: The comment close symbol, }, has been found on line *lineno*, character *charposition*, even though no comment has been opened.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Remove the close comment symbol if it is unnecessary or insert an open comment symbol where appropriate.

-2027 **Description of Error**: An illegal (invisible, control) character has been found on line *lineno*, character *charposition*. It has been replaced by a blank in the listing, but it is still in the source (input) table, and should be removed before attempting to compile again.

System Action Taken: The compile was successful.

Corrective Action: Remove the illegal character from the form specification file before attempting the next compile. If running under UNIX you can use the **od -c** *form-name* command to generate octal code that can help isolate the error. See the *UNIX Programmer's Manual* for more information about this command.

-2030 **Description of Error**: A typographical error has been found on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Edit the form specification file where indicated and correct the error.

-2032 **Description of Error**: The number above could not be successfully converted to either an INTEGER or a DOUBLE or a LONG.

System Action Taken: The compile was not completed.

Corrective Action: FORMBUILD could not convert the number provided. Acceptable LONG values are whole numbers between -2,147,483,647 and 2,147,483,647. Check that the number does not exceed these values (if a fixed point number) or that it does not contain an error (if a decimal number).

-2040 **Description of Error**: The table name *table-name* exceeds the maximum length of 10 characters.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name.*

-2041 **Description of Error**: The specification file *filename* cannot be opened. This is probably because the file does not exist.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *filename.*

-2044 **Description of Error**: A table needed by **FORMBUILD** is locked by another user.

System Action Taken: The compile was not completed.

Corrective Action: Wait a few minutes and try to recompile, or ask the user of the table to unlock the table. If you are certain no one is using the file, you can call up the RDSQL Menu and run an UNLOCK TABLE statement.

-2045 **Description of Error**: A table needed by **FORMBUILD** is exclusively locked by another user.

System Action Taken: The compile was not completed.

Corrective Action: Wait a few minutes and try to recompile, or ask the user of the table to unlock the table. If you are certain no one is using the file, you can call up the RDSQL Menu and run an UNLOCK TABLE statement.

-2800 **Description of Error**: The first line of the specification must be the keyword DATABASE followed by the database name.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the first line of the form specification file, or check that the keyword DATABASE is followed by the database name.

-2810 **Description of Error**: The name *database-name* is not an existing database name.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *database-name*. Check that *database-name* exists in your current directory or a directory included in your DBPATH environment variable.

-2811 **Description of Error**: The temporary table *table-name* could not be opened for writing.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system write permission to create the file. Contact your System Administrator if you need help with this action.

-2812 **Description of Error**: The temporary table *table-name* could not be read.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system read permission to access the file. Contact your System Administrator if you need help with this action.

-2820 **Description of Error**: The label name between brackets is incorrectly given or the label is missing.

System Action Taken: The compile was not completed.

Corrective Action: Check that the label name exists and is correctly spelled.

-2830 **Description of Error**: A left bracket has been found on this line, with no right bracket to match it.

System Action Taken: The compile was not completed.

Corrective Action: A set of brackets [ ] is used to delimit the field size of each field. Insert a right square bracket where appropriate into the form specification file. Note that a display field cannot be split across lines. For long character fields you can use subscripts to create more than one display field.

-2833 **Description of Error**: The database file *database-name* cannot be read by **FORMBUILD** to check the status of its indexes. Be sure that the user has permission to read this file and all directories along its path.

System Action Taken: The compile was not completed.

Corrective Action:

-2840 **Description of Error**: The label *label-name* was not defined in the form.

System Action Taken: The compile was not completed.

Corrective Action: Check that *label-name* is included in both the SCREEN and ATTRIBUTES sections of the form specification file, or delete the unnecessary *label-name*. (This error often accompanies errors -2820 and -2975.)

-2841 **Description of Error**: The form must include a tables declaration before the Attributes section.

System Action Taken: The compile was not completed.

Corrective Action: The form specification file must include DATABASE, SCREEN, TABLES, and ATTRIBUTES sections (in this sequence). Check the sequencing of these sections, and that you have included a TABLES section.

-2845 **Description of Error**: The composite column for table *table-name* containing column *column-name* is not indexed.

System Action Taken: The compile was not completed.

Corrective Action: All columns included in composites must be indexed in ascending order. Check that the column *column-name* is indexed. A composite index must be created containing all columns in the composite.

-2850 **Description of Error**: The name *column-name* is not a column name in this database.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the *column-name* column name.

-2853 **Description of Error**: The composite column member *column-name* is not joined to the proper column.

System Action Taken: The compile was not completed.

Corrective Action: Check the COMPOSITES statement in the INSTRUCTIONS section of the form. The identified *column-name* may be listed out of sequence.

-2854 **Description of Error**: The composite columns *column-name1* and *column-name2* do not have their individual member columns joined in the form.

System Action Taken: The compile was not completed.

Corrective Action: Check that all column members of the composite join are joined in the ATTRIBUTES section of the form.

-2856 **Description of Error**: The TODAY attribute may be assigned only to date columns.

System Action Taken: The compile was not completed.

Corrective Action: Check that the field in which the TODAY attribute has been used is a DATE column. If appropriate, remove the TODAY attribute from the ATTRIBUTES or INSTRUCTIONS section of the form.

-2857 **Description of Error**: There can be only one dominant composite column in a join list.

System Action Taken: The compile was not completed.

Corrective Action: Check that only one dominant composite (denoted with a leading asterisk *) appears in each join group. Edit the INSTRUCTIONS section and remove the asterisk from all but one of the groups in each join.

-2858 **Description of Error**: The table *table-name1* is joined to the table *table-name2*, but its joined columns do not belong to a composite key.

System Action Taken: The compile was not completed.

Corrective Action: You must first create a composite index for each set of joined columns in each table. Return to the RDSQL menu, save your statement, run a CREATE INDEX statement, and then reenter your original statement. (This error occurs when tables are joined by more than one column.)

-2859 **Description of Error**: The column *column-name* is a member of more than one table—you must specify the table name.

System Action Taken: The compile was not completed.

Corrective Action: The column name *column-name* appears in more than one table used in the form. You must specify the table from which *column-name* is to be accessed, using the format *table-name.column-name*.

-2860 **Description of Error**: There is a column/value type mismatch for *column-name.*

System Action Taken: The compile was not completed.

Corrective Action: Check that the value provided as the DEFAULT attribute or listed in the INCLUDE list matches the data type of the column (for example, DATE for a date column, or INTEGER for an integer).

-2861 **Description of Error**: You have exceeded the maximum of 20 tables.

System Action Taken: The compile was not completed.

Corrective Action: A maximum of 20 (this number may be larger on some systems) tables can be in use at any one time. When used by **PERFORM** your form would exceed this number. Reduce the number of tables included in the form.

-2862 **Description of Error**: The table *table-name* cannot be found in the database.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name.*

-2863 **Description of Error**: The column *column-name* does not exist among the specified tables.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the column name or check that *column-name* does exist in one of the specified tables.

-2864 **Description of Error**: The table *table-name* is not among the specified tables.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name* in the ATTRIBUTES and INSTRUCTIONS sections of the file. Check that the table is specified in the TABLES section of your form.

-2865 **Description of Error**: The column *column-name* does not exist in the table *table-name*.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *column-name*.

-2870 **Description of Error**: The subscripted column size does not match the space allocated in the display field.

System Action Taken: The compile was not completed.

Corrective Action: Check that the space provided in the display field is greater than or equal to the subscripted column size.

-2880 **Description of Error**: The word 'screen' has been left out.

System Action Taken: The compile was not completed.

Corrective Action: The keyword SCREEN must appear in all form specification files, and must begin in the first column on the line. Check the spelling and location of the SCREEN keyword or edit your statement to include it.

-2890 **Description of Error**: A screen definition must begin with a left brace {.

System Action Taken: The compile was not completed.

Corrective Action: Each screen layout must be enclosed within a pair of braces { }. Edit your statement to include the necessary left brace. Note that the left brace must appear in the first character position on the line.

-2892 **Description of Error**: The column *column-name* name appears more than once. If you wish a column to be duplicated in a form, use the same display field label.

System Action Taken: The compile was not completed.

Corrective Action: The column name *column-name* appears more than once in the ATTRIBUTES section of the form specification file. Use the same display field label (in the SCREEN section) to denote the repeated column, and remove any duplicate column names from the ATTRIBUTES section.

-2893 **Description of Error**: The display field label *field-tag* appears more than once in this form, but the lengths are different.

System Action Taken: The compile was not completed.

Corrective Action: A display field label may appear more than once in the SCREEN section, but in every instance the display fields must have identical lengths. Edit the field delimiters so that they are of equal length.

-2895 **Description of Error**: Display field length of number does not match the database column length of number. This is a warning only.

System Action Taken: The compile was completed.

Corrective Action: Check that the display field length (included in the SCREEN section) is equal to the table column size. (This error occurs only in character fields and with the **-v** option to **FORMBUILD**.)

-2920 **Description of Error**: The column *column-name* cannot be a dominant column because it is not indexed.

System Action Taken: The compile was not completed.

Corrective Action: You must first create an index for the column. (All columns included in joins must be indexed.) Return to the RDSQL menu and run a CREATE INDEX statement.

-2921 **Description of Error**: The database *database-name* is not compatible with the current version of **RDSQL**.

System Action Taken: The compile was not completed.

Corrective Action: The database was created under a prior version of **INFORMIX**. You must first convert the database, using **dbconvert,** before attempting to compile.

-2930 **Description of Error**: Portions of the column *column-name* are displayed on the screen more than once.

System Action Taken: The compile was not completed.

Corrective Action: Check the subscripting of *column-name* (present in the ATTRIBUTES section of the form specification file). Subscripts cannot overlap (for example, [25-49] and [50-75] are acceptable; [25-50] and [50-75] are unacceptable, as character 50 would have to appear twice).

-2931 **Description of Error**: There is an error in the format specification.

System Action Taken: The compile was not completed.

Corrective Action: You may only use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT or DATE column to control the format of the display. Check the format specifications for errors.

-2932 **Description of Error**: Formats may be specified only for DECIMAL, SMALLFLOAT, FLOAT or DATE columns.

System Action Taken: The compile was not completed.

Corrective Action: You may only use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT or DATE column to control the format of the display. Check that you have not specified the format on a CHAR, INTEGER, or SMALLINT column type.

-2933 **Description of Error**: The format width is larger than the allocated display width.

System Action Taken: The compile was not completed.

Corrective Action: Check that the format of a DECIMAL, SMALLFLOAT, FLOAT, or DATE column matches the length of the display field in the form.

-2934 **Description of Error**: The format width is less than the allocated display width. This is a warning only.

System Action Taken: The compile was completed.

Corrective Action: Check that the format of a DECIMAL, SMALLFLOAT, FLOAT, or DATE column matches the length of the display field in the form. (Note: Until this error is corrected any data displayed in the field may be truncated.)

-2940 **Description of Error**: The column *column-name* appears both with and without subscripts.

System Action Taken: The compile was not completed.

Corrective Action: Only column-names that are subscripted can appear more than once in the ATTRIBUTES section. Remove subscripts from *column-name* and all redundant references to it, or add subscripts to each.

-2941 **Description of Error**: *string-name* not a display field name on the form.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *string-name* (the *string-name* is a display field "field-tag" specified in a LET statement). Check that it is included in the SCREEN section of the form.

-2943 **Description of Error**: You have exceeded the pseudo machine capacity.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the complexity and/or number of the instructions in the form.

-2944 **Description of Error**: You may apply the "after add, update, query, or remove" commands to a table only—not a column.

System Action Taken: The compile was not completed.

Corrective Action: You cannot AFTER ADD, UPDATE, QUERY, or REMOVE a field. You may EDITADD and EDITUPDATE a field.

-2945 **Description of Error**: You may not calculate an aggregate on the display field *field-tag* because none of its associated database columns belong to the table *table-name*.

System Action Taken: The compile was not completed.

Corrective Action: You can only perform an aggregate on a display field associated with a specified table. Aggregates calculated in a control block statement can only be based on columns present in the table(s) cited in the control block statement.

-2946 **Description of Error**: You may not calculate an aggregate on the display-only field *field-tag*.

System Action Taken: The compile was not completed.

Corrective Action: You cannot perform any aggregates on a display-only field.

-2950 **Description of Error**: The column *column-name* has no section that starts at 1. Remember that the first subscript is one, not zero.

System Action Taken: The compile was not completed.

Corrective Action: Edit the form specification file so that the subscript to the column *column-name* begins with 1 (and not 0).

-2951 **Description of Error**: The left and right delimiters must be specified in a two-character string.

System Action Taken: The compile was not completed.

Corrective Action: When changing the delimiters that **PERFORM** uses to enclose the display fields in the SCREEN section of a form, you must specify both the left and right delimiters with one character each.

-2952 **Description of Error**: In order to use a picture, the picture length must be the same as the display field length.

System Action Taken: The compile was not completed.

Corrective Action: Edit the file so that the length of the picture specified with the PICTURE attribute equals the display field length in the SCREEN section.

-2953 **Description of Error**: The name *column-name* is not a database column in this form.

System Action Taken: The compile was not completed.

Corrective Action: All columns cited in a COMPOSITES list must be columns in the form. Check the spelling of the column name *column-name*.

-2955 **Description of Error**: The name *field-tag* is not a displayed field in this form.

System Action Taken: The compile was not completed.

Corrective Action: The display field *field-tag* has been specified in the ATTRIBUTES section of the form specification file, but the *field-tag* is not included in the SCREEN section of the form. Delete the *field-tag* from the ATTRIBUTES section or include it in the SCREEN section.

-2956 **Description of Error**: You may not set display field *field-tag* to a value because none of its associated database column(s) belong to the same table as *table-name*, nor is it a variable display-only field.

System Action Taken: The compile was not completed.

Corrective Action: You cannot assign values in a LET statement across tables. All associated fields to *field-tag* in the LET statement must belong to the same table as the control block table, or be display-only fields. Edit your LET statement.

-2957 **Description of Error**: You may not set the next field to *field-tag* because none of its associated database column(s) belong to the same table as *table-name*.

System Action Taken: The compile was not completed.

Corrective Action: You cannot set values in a LET statement across tables. You cannot set "nextfield=tag-name" to a tag-name associated with a column not in the table specified in the control block. Edit your LET statement.

-2958 **Description of Error**: You may have a maximum of 10 parameters in a C-function.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of parameters in the C-function to 10 or less.

-2959 **Description of Error**: Two tables may join with a maximum of 8 column pairs, including all components of composite columns.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of field joins to 8 or less.

-2970 **Description of Error**: The column *column-name* joins with other columns, but it is not indexed. **PERFORM** requires that join columns be indexed for cross-table queries.

System Action Taken: The compile was not completed.

Corrective Action: You must first index the column *column-name* using the **RDSQL** CREATE INDEX statement, and then recompile the form.

-2971 **Description of Error**: This column is not a character column, and therefore cannot be subscripted.

System Action Taken: The compile was not completed.

Corrective Action: Remove subscripting from any non-CHARACTER columns in your form.

-2972 **Description of Error**: This column cannot be right justified or zero-filled because its displayed width does not match the actual column width.

System Action Taken: The compile was not completed.

Corrective Action: The display field width must be greater than or equal to the actual column width. Edit the display field size.

-2973 **Description of Error**: There may be only one dominant column in a display field description.

System Action Taken: The compile was not completed.

Corrective Action: You can only have one dominant column (denoted with a leading asterisk *) in any single join. Edit the join to include only one dominant column.

-2975 **Description of Error**: The display field label *field-tag* has not been used.

System Action Taken: The compile was not completed.

Corrective Action: A display field label *field-tag* present in the SCREEN section of the form specification file does not correspond to a column in the ATTRIBUTES section. Delete the display field label *field-tag* if it is unnecessary or include a reference to it in the ATTRIBUTES section.

-2977 **Description of Error**: Table *table-name1* cannot be a master of table *table-name2* because they do not join.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of each table name and that at least one column in table *table-name1* is joined to a column in *table-name2*.

-2978 **Description of Error**: The column *column-name1* and the column *column-name2* cannot be joined columns because their types or lengths are different.

System Action Taken: The compile was not completed.

Corrective Action: Columns included in a join must have matching data types and lengths. Check that the columns *column-name1* and *column-name2* are of the same type and length.

-2986 **Description of Error**: The form specification has exceeded the maximum of 20 master/detail pairs.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of master/detail pairs present in the form specification file to 20 or less.

-2987 **Description of Error**: The form specification has exceeded the maximum of 20 screens.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of screens included in the form specification file to 20 or less.

-2988 **Description of Error**: **FORMBUILD** has run out of memory.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the data space limit on your machine. Reduce the complexity of the form specification file.

-2989 **Description of Error**: The column *column-name* is a reference column, but it is not indexed. **PERFORM** requires that reference columns be indexed for lookups.

System Action Taken: The compile was not completed.

Corrective Action: First index the column, then recompile the form.

-2990 **Description of Error**: The columns *column-name1* and *column-name2* are not in the same database table.

System Action Taken: The compile was not completed.

Corrective Action: All columns cited in a single LOOKUP attribute must belong to the same table.

-2992 **Description of Error**: The display label *field-tag* has already been used.

System Action Taken: The compile was not completed.

Corrective Action: Each display label tag must be unique. Select a new display label tag.

-2993 **Description of Error**: There is a circular join path specified in the form.

System Action Taken: The compile was not completed.

Corrective Action: You cannot include a series of column joins that have a net effect of "**table1.columnA** joins **table2.columnA**, **table2.columnB** joins **table3.columnB**, and **table3.columnC** joins **table1.columnC**." (A single join of "**table1.columnA** joins **table2.columnA** joins **table3.columnA**" is allowed.) Remove one of the joins so that this circular join path no longer exists.

-2994 **Description of Error**: The form has exceeded the maximum number of joins between tables.

System Action Taken: The compile was not completed.

Corrective Action: The form exceeds the maximum number (8) of tables included in joins. Reduce the total number of tables included in joins.

-2995 **Description of Error**: The form has exceeded the maximum number of tables contained in joins.

System Action Taken: The compile was not completed.

Corrective Action: The form exceeds the maximum number of tables included in joins. Reduce the total number of tables included in joins.

-2996 **Description of Error**: The unanticipated error number *error-number* has occurred.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: *error-number* is an operating system error number. If running under UNIX, check the *UNIX Programmer's Manual* for error information relating to *error-number*. Contact your System Administrator if you need assistance with this action. You might also contact the Technical Support Department of the company that supplied your operating system for advice about this error number.

-2997 **Description of Error**: See error number *errno*.

System Action Taken: The compile was not completed.

Corrective Action: Locate the indicated error message in this section.

-2998 **Description of Error**: Operating system error *error-number*: *string*.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: If the error number is between 1-99 and you are using a UNIX-based system, check your *UNIX Programmer's Manual* for the error corresponding to the number indicated. Contact your System Administrator if you need assistance with this action. If you are using a DOS system, call RDS Technical Support.

-3000 **Description of Error**: C-ISAM or system error number *errno*.

System Action Taken: The compile was not completed.

Corrective Action: If the error number is between 1-99 and you are using a UNIX-based system, check your *UNIX Programmer's Manual* for the error corresponding the number indicated. Contact your System Administrator if you need assistance with this action. If you are using a DOS system, call the Technical Support Department at RDS. If you receive a C-ISAM error number (100-199), locate the error message in this appendix.

-3001 **Description of Error**: There are no forms in the current directory.

System Action Taken: PERFORM can not locate a form.

Corrective Action: You can only run forms present in the current directory or a directory accessed in your DBPATH environment variable. Change directories to the one containing the desired form(s), or check the setting of your DBPATH environment variable. Contact your System Administrator if you need help with this action.

-3005 **Description of Error**: You do not have permission to access the form *form-name*.

System Action Taken: The statement was not processed (operating system error).

Corrective Action: Check that you have operating system read and write permission to the file *form-name.frm*. Contact your System Administrator if you need help with this action.

-3007 **Description of Error**: Your form is incompatible with the current version of **PERFORM**. Please use **FORMBUILD** to re-build it, and then run **PERFORM** again.

System Action Taken: The form was not accessible to **PERFORM**.

Corrective Action: Use **FORMBUILD** to re-build the form, and then run **PERFORM** again.

-3010 **Description of Error**: Database *database-name* not found.

System Action Taken: **PERFORM** could not locate the database specified in the form.

Corrective Action: Check the spelling of *database-name* or check that *database-name* resides in your current directory (the database will be listed as *database-name.dbs*), or a directory contained in your DBPATH environment variable. Check that you have permission to access the database.

-3011 **Description of Error**: Operating system error *error-number*: *string*.

System Action Taken: The statement containing the error was not processed (operating system error).

Corrective Action: If the error number is between 1-99 and you are using a UNIX-based system, check your *UNIX Programmer's Manual*. If you are using a DOS system, call the Technical Support Department at RDS. Contact your System Administrator if you require assistance with these actions.

-3012 **Description of Error**: Duplicate row.

System Action Taken: The statement was not processed.

Corrective Action: Check all unique index columns for duplicate values.

-3013 **Description of Error**: File not open.

System Action Taken: The statement was not processed.

Corrective Action: Check that the audit trail file for the indicated *table-name* has not been removed. You may need to start another audit trail file.

-3017 **Description of Error**: This row or the entire table is locked.

System Action Taken: The requested action was not completed.

Corrective Action: Two options:

1. Another individual may be using the row or table. Wait a few moments and reenter your command.

2. Check that the table is not currently being locked by someone else. Contact the user of the table and ask this individual to run an RDSQL UNLOCK TABLE statement. If you are certain that the table is not in use, you may need to empty (NOT remove) the contents of the *tablename.lok* file. (This file contains information about which rows of the table are being used at any one time, and whether the entire table is locked.) You can copy the "file" **/dev/null** into this file to empty the file. Contact your System Administrator about this action.

-3018 **Description of Error**: A table is locked—no reading or writing is permitted.

System Action Taken: The requested action was not completed.

Corrective Action: Another user has locked the table in exclusive mode. Contact this individual and ask that the table be unlocked.

-3020 **Description of Error**: The table *tablename-name* is not in the database—build your form again.

System Action Taken: The requested action was not completed.

Corrective Action: **PERFORM** cannot access the table. Check that the table has not been removed from the database.

-3030 **Description of Error**: The column *column-name* is not in the database—build your form again.

System Action Taken: The requested action was not completed.

Corrective Action: **PERFORM** cannot access the column. Check that the column has not been removed from the table.

-3031 **Description of Error**: The form *form-name* does not exist.

System Action Taken: The requested action was not completed.

Corrective Action: Check the spelling of the form name. The form may not have been created or may have been removed. Check that the form exists in your current directory or a directory included in your DBPATH environment variable.

-3035 **Description of Error**: The column *column-name* has changed column type—build your form again.

System Action Taken: The requested action was not completed.

Corrective Action: **PERFORM** cannot access the column. Someone has executed a **RDSQL** command to change the data type of the column. Rebuild the form.

-3037 **Description of Error**: The column *column-name* is not a composite column—build your form again.

System Action Taken: The requested action was not completed.

Corrective Action: Check that all columns included in the composites are indexed as composites—someone may have removed a composite index since the form was compiled.

-3050 **Description of Error**: Operating system error. Cannot open a temporary file.

System Action Taken: The requested action was not completed (operating system error).

Corrective Action: Two options:

(1) Check that you have operating system permission to read and write a file in the **/tmp** directory. Contact your System Administrator if you assistance help with this action.

(2) You may have exceeded the operating system open file limit. If this is the case you will need to reduce the number of tables included in your form.

-3060 **Description of Error**: Operating system error. Cannot create a temporary file.

System Action Taken: The requested action was not completed (operating system error).

Corrective Action: Several possible situations can invoke this message:

(1) Check that you have operating system read and write permission to the **/tmp** directory. Contact your System Administrator if you need assistance with this action.

(2) Check that a file with the same name does not already exist in the **/tmp** directory. If it does, rename or delete it.

(3) You may have exceeded the operating system open file limit. If this is the case you will need to reduce the number of tables included in your form.

(4) The **/tmp** directory may be full. Ask your System Administrator to remove any unneeded files.

(5) If you are on a DOS system, your disk may be full. Remove any unneeded files from the disk.

-3080 **Description of Error**: Cannot create unique row identifier.

System Action Taken: The requested action was not completed.

Corrective Action: The C-ISAM file has been corrupted. Run the **bcheck** utility on the table you are attempting to add to.

-3081 **Description of Error**: The specified output file name has a bad extension.

System Action Taken: The requested action was not completed.

Corrective Action: An output filename cannot have the extensions **.dat .idx .lok .aud .per .frm .ace .arc .sql** or **.unl**. Select a different filename.

-3100 **Description of Error**: Row with a duplicate unique key exists already.

System Action Taken: The requested action was not completed.

Corrective Action: You have entered a value into a field that does not allow duplicates (it is indexed as a unique column). Select an alternate value or drop the unique index.

-3101 **Description of Error**: The column *column-name* does not allow NULL values.

System Action Taken: The requested action was not completed.

Corrective Action: A column requiring input has not been given data when adding data to a table. You must provide a value for the identified field.

-3120 **Description of Error**: Error in field.

System Action Taken: The action failed.

Corrective Action: Check that the data type of the data entered into the field is appropriate for that field.

-3200 **Description of Error**: There are no rows in the current list.

System Action Taken: The action failed.

Corrective Action: You must first query the table for row data before executing UPDATE, REMOVE, NEXT, or PREVIOUS command.

-3210 **Description of Error**: The current row position contains a deleted row.

System Action Taken: The request was not completed.

Corrective Action: The current row has been removed by another user. You cannot use the UPDATE, REMOVE, NEXT, or PREVIOUS commands with a non-existing row.

-3260 **Description of Error**: This is an invalid value—it does not exist in *table-name* table.

System Action Taken: The request was not completed.

Corrective Action: The column has a verify join with a column in the *table-name* table. Any value entered in the column must correspond to an existing value in the column in the other table.

-3261 **Description of Error**: Invalid value—its composite value does not exist in *table-name* table.

System Action Taken: The request was not completed.

Corrective Action: The column is part of a composite verify join with table *table-name*. Any value entered in the field must correspond to an existing (composite) value in the *table-name* table.

-3265 **Description of Error**: A required index on *table-name* is missing—please restore.

System Action Taken: The request was not completed.

Corrective Action: All dominant columns (those with asterisks before their names) in verify joins must be indexed. Index the column(s) and then reenter **PERFORM**.

-3266 **Description of Error**: A required composite index containing *column-name* must be restored.

System Action Taken: The request was not completed.

Corrective Action: You must create a composite index for the columns involved in the composite joins. Return to the RDSQL menu and run a CREATE INDEX statement.

-3300 **Description of Error**: There are no more rows in the direction you are going.

System Action Taken: The request was not completed.

Corrective Action: You have reached the beginning or end of the current list. Use the NEXT or PREVIOUS command to reverse the direction of the display of the rows present in the current list.

-3400 **Description of Error**: There are no rows satisfying the conditions.

System Action Taken: The query was completed.

Corrective Action: The query has identified no matching rows. Enter a new query.

-3420 **Description of Error**: This value is not among the valid possibilities.

System Action Taken: The entered value has been rejected.

Corrective Action: A list or range of acceptable values has been established for this column (via an INCLUDE attribute in the form specification file). You must enter a value within the acceptable range.

-3430 **Description of Error**: This field requires an entered value.

System Action Taken: The request was not completed.

Corrective Action: The row cannot be added to the table until a value has been entered in the indicated field. (A REQUIRED attribute in the form specification file has been associated with the column.) Enter a value in the field.

-3450 **Description of Error**: Someone else has deleted a row that is in your list.

System Action Taken: The requested action was not completed.

Corrective Action: Another individual is accessing a table included in the form you are using and has deleted a row in your current list.

-3460 **Description of Error**: This row has been locked by another user—try again later.

System Action Taken: The requested action was not completed.

Corrective Action: The table is currently being used by another individual. Wait a few moments and reenter your statement.

-3500 **Description of Error**: The two entries were not the same—please try again.

System Action Taken: No value was entered in the field.

Corrective Action: The row cannot be added to the table until the value in the field is entered twice. (A VERIFY attribute in the form specification file has been associated with the column.) The values entered must be identical. Carefully reenter each.

-3600 **Description of Error**: No detail table has been specified for this table.

System Action Taken: The requested action cannot be completed.

Corrective Action: You must first establish this table as the master of another table in the INSTRUCTIONS section of the form.

-3610 **Description of Error**: No master table has been specified for this table.

System Action Taken: The requested action cannot be completed.

Corrective Action: You must first establish this table as the detail of another table in the INSTRUCTIONS section of the form.

-3620 **Description of Error**: You do not have permission to write into this table.

System Action Taken: The requested action cannot be completed.

Corrective Action: You do not have insert or update privilege for the table and so can not insert or update a column. Check with the owner of the table or your Database Administrator for the necessary permission.

-3630 **Description of Error**: Cannot write into a temporary file.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Another user (with operating system super-user privileges) has deleted the temporary file after you began working with it.

-3650 **Description of Error**: You must first remove *tablename* rows that join this row.

System Action Taken: The requested action cannot be completed.

Corrective Action: The row you wish to remove is the dominant member of a verify join. You cannot remove this row without first removing any rows that join it.

-3651 **Description of Error**: You have exceeded the number of opened tables allowed by your system.

System Action Taken: The requested action failed (operating system limitation).

Corrective Action: Executing the statement would require that **PERFORM** open more files than allowed by your operating system. Reduce the number of tables included in your form.

-3660 **Description of Error**: You must first update *table-name* rows that join this row.

System Action Taken: The requested action failed.

Corrective Action: The row you wish to update is included as the dominant column in a verify join. You are not allowed to change a row value in this column that would invalidate joined rows in other tables. You must first update the rows in *table-name* that join this column.

-3670 **Description of Error**: Cannot seek into a temporary file.

System Action Taken: Your statement did not run (operating System error).

Corrective Action: A temporary file used by you while in **PERFORM** has been removed. Exit from **PERFORM** and then reenter **PERFORM** and continue your work.

-3680 **Description of Error**: PERFORM has run out of memory.

System Action Taken: The requested action was not completed.

Corrective Action: PERFORM has run out of data space. Reduce the complexity of your form.

-3681 **Description of Error**: Your terminal type is unknown to the system.

System Action Taken: The requested action was not completed.

Corrective Action: Check the setting of the TERM and TERMCAP environment variables. Make sure an entry for your terminal type is present in the **termcap** file. Contact your System Administrator if you need help with these actions.

-3690 **Description of Error**: Detail cannot be executed—tables *table-name1* and *table-name2* do not join.

System Action Taken: The requested action cannot be completed.

Corrective Action: A master-detail relationship for two tables has been specified in the INSTRUCTIONS section for the form, but the tables do not join. Add a join to the ATTRIBUTES section.

-3700 **Description of Error**: Permission not granted to allow reading of *table-name* or *column-name.*

System Action Taken: Your request has been denied.

Corrective Action: You are not granted permission to retrieve data from the table or column. Request permission from the table owner or your Database Administrator.

-3710 **Description of Error**: Permission not granted to allow update of *table-name* or *column-name*.

System Action Taken: Your request has been denied.

Corrective Action: You are not granted permission to update rows in the table. Request permission from the table owner or your Database Administrator.

-3720 **Description of Error**: Permission not granted to allow add of *table-name* or *column-name*.

System Action Taken: Your request has been denied.

Corrective Action: You are not granted permission to insert rows into the table. Request permission from the table owner or your Database Administrator.

-3730 **Description of Error**: Permission not granted to allow remove of *table-name* row.

System Action Taken: Your request has been denied.

Corrective Action: You are not granted permission to delete rows in the table. Request permission from the table owner or your Database Administrator.

-3731 **Description of Error**: Cannot open the table *table-name*.

System Action Taken: The requested action cannot be completed.

Corrective Action: Two possibilities: (1) Check that you have operating system read permission to access the file. (3) You may have exceeded the operating system open file limit. If this is the case you will need to reduce the number of tables included in your form.

-3732 **Description of Error**: The database *database-name* is not compatible with the current version of INFORMIX-SQL.

System Action Taken: The requested action cannot be completed.

Corrective Action: The database was generated using a prior version of INFORMIX or INFORMIX-SQL. If you are using a INFORMIX database, you must first convert the database with the **dbconvert** program before using it with INFORMIX-SQL. If you are using a database created under a prior version of INFORMIX-SQL you must update with the **dbupdate** program before using it with INFORMIX-SQL.

-3752 **Description of Error**: You cannot use this editing feature because a picture exists.

System Action Taken: The PERFORM action failed.

Corrective Action: You cannot use certain PERFORM editing actions (for example, CTRL-A, CTRL-D, and CTRL-X) while updating or adding data to a field in which a PICTURE attribute has been assigned.

-3754 **Description of Error**: You have exceeded the pseudo machine stack space.

System Action Taken: The PERFORM action failed.

Corrective Action: Reduce the complexity of the form (in particular, simplify the control block statements). A complex LET statement should be split into several simpler LET statements.

-3755 **Description of Error**: The selected display-only field *field-tag* cannot be accessed.

System Action Taken: The **PERFORM** action failed.

Corrective Action: pf__nxfield cannot select a display-only field.

-3756 **Description of Error**: The selected field *field-tag* cannot be assigned a value at this time.

System Action Taken: The **PERFORM** action failed.

Corrective Action: pf__putval cannot assign a value to a *field-tag* that is not in the current file. Change your user function.

-3758 **Description of Error**: *field-tag* is not in the current table.

System Action Taken: The **PERFORM** action failed.

Corrective Action: The *field-tag* specified for pf-nxfield is not found in the current table. Revise your user function.

-3759 **Description of Error**: The display field *field-tag* does not exist in the form.

System Action Taken: The compile was not completed.

Corrective Action: pf__getval, pf__putval, or pf__nxfield has referenced a *field-tag* that is not in the form. Revise your user function.

-3761 **Description of Error**: ERROR—attempt to divide by zero has occurred.

System Action Taken: The request was not completed.

Corrective Action: A calculation in the INSTRUCTIONS section requires **PERFORM** divide by zero. Check whether a LET statement requires a value be divided by zero.

-3762 **Description of Error**: The user function *function-name* defined in the form could not be found in the C function definition table **userfuncs**.

System Action Taken: The request was not completed.

Corrective Action: You must compile your C program and create the compiled "newperf" program file. You can then run the form using the "newperf" program in place of **PERFORM**.

-8001 **Description of Error**: The number of newlines specified for TOP MARGIN, FIRST PAGE HEADER (PAGE HEADER), PAGE TRAILER, and BOTTOM MARGIN must leave enough room on the page for the printing of the ON EVERY RECORD clause.

System Action Taken: The compile was not completed.

Corrective Action: The total number of lines required by all the above statements exceeds the page length. Check that you have not added an unwanted digit to one of the newline requests (for example, 30 instead of 3), or incorrectly set the page length. Increase the page length or reduce the number of lines printed in all of the above statements.

-8002 **Description of Error**: The ACE report specification is too complex or large to be properly compiled. The number of instructions needed to implement this specification is greater than the space allocated in the compiler's instruction tables.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of instructions contained in the report.

-8003 **Description of Error**: An INTEGER constant may not be larger than 32,767.

System Action Taken: The report was not compiled.

Corrective Action: Check the value of the SMALLINT constant, and that this value is a whole number. (Allowable SMALLINT ranges are -32,767 to 32,767.) Characters other than digits are not acceptable.

-8004 **Description of Error**: Illegal DOUBLE constant.

System Action Taken: The report was not compiled.

Corrective Action: Check the value of the FLOAT constant. Characters other than digits are not acceptable.

-8006 **Description of Error**: A quoted string may not exceed 80 characters in length.

System Action Taken: The report was not compiled.

Corrective Action: Reduce the length of the quoted string to 80 characters or less.

-8008 **Description of Error**: The file *filename* could not be opened. The operating system was asked to open it for reading.

System Action Taken: The report was not compiled (operating system error).

Corrective Action: Check the spelling of *filename*. Check that you have operating system read permission to read the file. Contact your System Administrator if you need help with this action.

-8009 **Description of Error**: A memory allocation error has occurred. The ACE report specification is probably too large.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the available data space on your machine. Reduce the length or complexity of your ACE report.

-8010 **Description of Error**: The file *filename* could not be opened. The operating system was asked to open it for writing. This is an internal system error.

System Action Taken: The report was not compiled (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the current directory, the **/tmp** directory, or a directory cited in your DBTEMP environment variable. Contact your System Administrator if you need help with this action.

-8013 **Description of Error**: An error occurred when an attempt was made to open the listing file *filename*. Be certain that the user has write permissions in the current directory.

System Action Taken: The report was not compiled.

Corrective Action: Check that you have operating system write permission in the current directory. Contact your System Administrator if you need help with this action.

-8014 **Description of Error**: Usage: SACEPREP [**-q**] **-o** *output_directory*] *input_filename*

flags:

-q SACEPREP will not print any messages or errors to the screen.

-o SACEPREP will take the next argument as the directory where the **.err** and **.arc** files will be written.

System Action Taken: The request was not completed.

Corrective Action: Carefully retype the command line, noting allowable flags and required arguments.

-8015 **Description of Error**: An open comment symbol, {, was found inside an already open comment on line *line-number*, character *character-position*. This could be due to a failure to close the previously opened comment, which was begun on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Insert a close comment symbol where appropriate in the report specification file. (Note: You cannot nest comments.)

-8016 **Description of Error**: A comment has been opened, but not closed. The last comment begun was opened on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Insert a close comment symbol where appropriate in the report specification file.

-8017 **Description of Error**: An illegal (invisible, control) character has been found on line *lineno*, character *charposition*. It has been replaced by a blank in the listing, but it is still in the source (input) file, and should be removed before attempting to compile again.

System Action Taken: The compile was not successful.

Corrective Action: Remove the illegal character from the form specification file before the next compile.

-8018 **Description of Error**: A grammatical error has been found on line *lineno*, character *charposition*. The construct is not understandable in its context.

System Action Taken: The report was not compiled.

Corrective Action: A syntax error exists in the source file. A keyword may be misspelled or misplaced in the file. Check the spelling of the keyword or the sequence of control blocks within the report specification file.

-8021 **Description of Error**: The source file *filename* cannot be opened. This is probably because the file does not exist.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *filename*. Check that the report specification file exists in your current directory or that the correct pathname has been provided.

-8022 **Description of Error**: This identifier exceeds the maximum length for identifiers, which is 50.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the length of the identifier.

-8025 **Description of Error**: The comment close symbol } has been found on line *lineno*, character *charposition*, even though no comment has been opened.

System Action Taken: The compile was not completed.

Corrective Action: Insert an open comment symbol where appropriate in the report specification file or remove the unnecessary close comment symbol.

-8027 **Description of Error**: An illegal (invisible, control) character has been found on line *lineno*, character *charposition*. It has been replaced by a blank in the listing, but it is still in the source (input) file, and should be removed before attempting to compile again.

System Action Taken: The compile was not successful.

Corrective Action: Remove the illegal character from the form specification file before the next compile.

-8030 **Description of Error**: A typographical error has been found on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Check that an illegal character has not been placed at that point in your file.

-8033 **Description of Error**: The source file name *filename* exceeds the maximum length of 10.

System Action Taken: The compile was not completed.

Corrective Action: Check the length of the filename. Rename the file if necessary.

-8034 **Description of Error**: An unknown flag was found on the command line.

System Action Taken: The requested action was not completed.

Corrective Action: Carefully retype the command line. Allowable ACEPREP flags are **-q** and **-o**; allowable ACEGO flags are **-q** and **-d**.

-8051 **Description of Error**: The LEFT MARGIN must be no less than zero and no greater than the RIGHT MARGIN.

System Action Taken: The compile was not completed.

Corrective Action: Check that the LEFT MARGIN specification is not less than zero or greater than the value of the RIGHT MARGIN specification.

-8053 **Description of Error**: Neither the PAGE LENGTH, TOP MARGIN, nor BOTTOM MARGIN is allowed to be less than zero.

System Action Taken: The compile was not completed.

Corrective Action: Check that the PAGE LENGTH, TOP MARGIN, or BOTTOM MARGIN specification has not been set to less than zero.

-8054 **Description of Error**: The string *string* does not fit into remaining space in the compiler's string table.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the length of strings and/or the number of unique strings included in the report. (Note: identical strings are stored only once.)

-8055 **Description of Error**: This constant causes the constant table in ACEPREP to overflow.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of numeric constants used in the report specification file.

-8056 **Description of Error**: Skipping lines inside of WHILE or FOR loops is not allowed within PAGE HEADERS or TRAILERS.

System Action Taken: The compile was not completed.

Corrective Action: The number of lines devoted to the page header and page trailer must remain unchanged within the report. Skipping lines inside of WHILE or FOR loops within PAGE HEADERS or TRAILERS would violate this rule.

-8057 **Description of Error**: Only user variables of type CHARACTER may be subscripted or have lists in the LET statement.

System Action Taken: The compile was not completed.

Corrective Action: You can subscript on CHAR data types only. In a LET statement non-CHAR types can have only one user-assigned value. A list of character strings can be assigned only to a CHAR type user variable.

-8059 **Description of Error**: The limit for "IF" statement nesting has been exceeded. There are too many "IF" statements within "IF" statements for ACE to properly compile this specification.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the maximum of 128 levels of nested "IF" statements. Remove one or more statements.

-8102 **Description of Error**: There may be only one AFTER or BEFORE GROUP OF clause for any single column specified in the ORDER BY command.

System Action Taken: The compile was not completed.

Corrective Action: Only one BEFORE GROUP OF and/or AFTER GROUP OF clause is allowed for each single column specified in the ORDER BY command. You must collapse any multiple clauses for the same column into a single clause.

-8103 **Description of Error**: In order for the BEFORE or AFTER GROUP OF clause to function properly, the column specified in the clause must also have been specified as a sort column in the ORDER BY command. If the sort column is subscripted, the column in the BEFORE or AFTER GROUP OF clause must be also.

System Action Taken: The compile was not completed.

Corrective Action: Include the column name specified in the BEFORE or AFTER GROUP OF clause in the ORDER BY clause in the SELECT statement. Subscript the column name where present in the BEFORE or AFTER GROUP OF clause.

-8104 **Description of Error**: Group aggregates can only be used in an AFTER GROUP OF clause.

System Action Taken: The compile was not completed.

Corrective Action: Remove any group aggregates from non-AFTER GROUP OF clauses.

-8105 **Description of Error**: Aggregates may not be used within another aggregate. Nor may aggregates be used within the WHERE clause of another aggregate.

System Action Taken: The compile was not completed.

Corrective Action: Aggregates can not be nested. Remove any aggregates placed within an aggregate. Remove any aggregates within the WHERE clause of any other aggregate.

-8107 **Description of Error**: A user variable or parameter has been defined by the user more than once.

System Action Taken: The compile was not completed.

Corrective Action: Each user variable or parameter must have a unique name. Select an alternate name for the variable or parameter.

-8108 **Description of Error**: A user variable or parameter must have a length greater than zero.

System Action Taken: The compile was not completed.

Corrective Action: Check that a CHAR or DECIMAL field has a length greater than zero.

-8110 **Description of Error**: The user variable called *variable-name* has not yet been defined by the user.

System Action Taken: The compile was not completed.

Corrective Action: Include a definition for the variable in the DEFINE section of the report.

-8111 **Description of Error**: User variables and runtime parameters may not be used in an ORDER BY statement.

System Action Taken: The compile was not completed.

Corrective Action: You can only ORDER BY database columns. Remove any user variables or runtime parameters present in your ORDER BY statement.

-8112 **Description of Error**: Within an IF-THEN-ELSE statement of a header or trailer clause the number of lines printed in the IF part must equal the number of lines printed in the ELSE part.

System Action Taken: The compile was not completed.

Corrective Action: The number of lines printed in each header or trailer cannot vary from page to page. An IF THEN ELSE statement with the potential to generate an uneven number of lines would violate this rule. Rewrite the clause to balance the number of lines printed in each part of the statement.

-8113 **Description of Error**: SKIP TO TOP OF PAGE is not allowed in any header or trailer clause.

System Action Taken: The compile was not completed.

Corrective Action: Remove the SKIP TO TOP OF PAGE statement from the header or trailer clause. (A SKIP TO TOP OF PAGE statement in the header clause could generate an infinite number of pages with headers only.)

-8114 **Description of Error**: The number of lines to be printed in the top and bottom margins plus the lines to be printed in the page header and trailer clauses exceed the page length.

System Action Taken: The compile was not completed.

Corrective Action: Check that you have not made a typing error in denoting the size of the top or bottom margins, the page length, or the number of lines to skip in the page header or trailer control blocks.

-8115 **Description of Error**: There may be no more than 8 sort columns specified in the ORDER BY command.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of sort columns to 8 or less.

-8116 **Description of Error**: Illegal subscripting in ORDER BY command.

System Action Taken: The compile was not completed.

Corrective Action: You may only subscript a character field. Check the numerical sequence of subscripts. There can be no negative numbers, and the left number must be less than or equal to the right number. ([1,6] is allowed; [8,3] is not allowed.)

-8122 **Description of Error**: NEED n LINES may not be used in PAGE HEADER or PAGE TRAILER clauses.

System Action Taken: The compile was not completed.

Corrective Action: A NEED n LINES statement is unnecessary in the PAGE HEADER or PAGE TRAILER clauses. Remove the statement from the report specification file.

-8123 **Description of Error**: The PRINT FILE command is not allowed within the FIRST PAGE HEADER, PAGE HEADER, or PAGE TRAILER clauses.

System Action Taken: The compile was not completed.

Corrective Action: Remove the PRINT FILE command from the FIRST PAGE HEADER, PAGE HEADER, or PAGE TRAILER clauses in the report specification file.

-8124 **Description of Error**: A user variable used in the above query has not been defined by the user in the DEFINE section.

System Action Taken: The compile was not completed.

Corrective Action: A variable referred to in the SELECT statement has not been defined in the DEFINE section of the report. Define the variable.

-9001 **Description of Error**: Only columns of type CHARACTER may be subscripted or printed without trailing blanks.

System Action Taken: The report was not generated.

Corrective Action: Check that you have not attempted to subscript or clip a non-CHAR field.

-9002 **Description of Error**: An unknown operation code was found in the PMIM.

System Action Taken: The report was not generated.

Corrective Action: An internal ACEGO error has occurred. Contact RDS Technical Support for assistance.

-9004 **Description of Error**: The column *column-name* is not in the current table.

System Action Taken: The report was not generated.

Corrective Action: Check the spelling of the column name. Check that the column name is included in the SELECT list of the last SELECT statement in the report. Also, make sure you have not prepended a temporary table name to a column name used in the FORMAT section. For example, *temporary-table.column-name* is illegal in the FORMAT section.

-9014 **Description of Error**: Usage: SACEGO [**-q**] **-d** *database*] *report_filename*

flags:

-q SACEGO will not print any messages or errors to the screen.

-d SACEGO will use the database name that follows the **-d** option instead of the database name specified in the DATABASE section of the report specification.

System Action Taken: The requested action was not completed.

Corrective Action: Carefully retype the command line, noting the allowable flags and required arguments.

-9025 **Description of Error**: The column *column-name* is not in the current table. All columns used as an ORDER BY column must be in the current table.

System Action Taken: The report was not generated.

Corrective Action: The column name is not accessible to the current SELECT statement. Only columns present in the current SELECT statement may be used in the ORDER BY clause. You may have included a column present in a prior SELECT statement in the ORDER BY clause.

-9046 **Description of Error**: The report output file *filename* could not be opened. This is probably due to permissions.

System Action Taken: The request was not completed (operating system error).

Corrective Action: Check that you have operating system write permission to write in the selected directory or that you have not exceeded the operating system open file limit of 20 files. Contact your System Administrator if you need help with these actions.

-9047 **Description of Error**: The output file *filename* could not be created.

System Action Taken: The report was not generated (possible operating system error).

Corrective Action: The file *filename* may already exist. Select an alternate name for the output file. Check that you have operating system write permission to write in the selected directory. Check that sufficient disk space is available for the report. Contact your System Administrator if you need help with these actions.

-9048 **Description of Error**: The pipe *program-name* could not be opened.

System Action Taken: The report was not generated. (an operating system error).

Corrective Action: Check that you have not misspelled the name of the program to receive the output, or contact your System Administrator.

-9049 **Description of Error**: A value entered by the user as a command line parameter cannot be converted to the data type *data-type.*

System Action Taken: The report was not generated.

Corrective Action: The value entered cannot be converted to the specified data type. Check that you have not included a CHAR data type where a NUMERIC type is required, or a CHAR in place of a DATE, etc.

-9050 **Description of Error**: The user variable *variable-name* is of type *data-type.* The value entered by the user cannot be converted to that type.

System Action Taken: The report was not generated.

Corrective Action: The value entered cannot be converted to the specified data type. Check that you have not included a CHAR data type where a NUMERIC type is required, or a CHAR in place of a DATE, etc.

-9051 **Description of Error**: The number of defined parameters in the ACE program does not equal the number of actual parameters passed to ACEGO on the command line.

System Action Taken: The report was not generated.

Corrective Action: The user has specified a number of parameters inside the DEFINE section of the specification. The number of parameters actually entered at the command line does not equal the number of parameters defined in the specification. Parameters that contain blanks should be enclosed within quotation marks.

-9052 **Description of Error**: The format string is too large. It may not be larger than 79 characters.

System Action Taken: The report was not generated.

Corrective Action: Reduce the length of the format string to 79 characters or less.

-9054 **Description of Error**: A version incompatibility has been detected. Recompile your ACE report specification using ACEPREP, and then rerun ACEGO. Be certain that the version numbers for ACEPREP and ACEGO are identical.

System Action Taken: The report was not generated.

Corrective Action: You are attempting to use an ACE report compiled under a prior version of ACEPREP. Recompile your report specification file using ACEPREP and then rerun ACEGO. Be certain that the version numbers for ACEPREP and ACEGO are identical.

-9055 **Description of Error**: The ACE Report Control file *filename* could not be found.

System Action Taken: The report was not generated.

Corrective Action: ACEGO expects to locate the control file *filename.arc* in your current directory or a directory cited in your DBPATH environment variable. ACEGO has been unable to find the file in either location. Check your DBPATH environment variable. Contact your System Administrator if you need help with this action.

-9060 **Description of Error**: The values *m* and *n* used to subscript *column-name* are outside the bounds for the column's defined length.

System Action Taken: The report was not generated.

Corrective Action: Check that you have not provided illegal subscript values to the column. (For example, [10,40] is an illegal subscript for a CHAR field of length 20.)

-9062 **Description of Error**: An error has occurred during the conversion of a numeric value to a CHARACTER type user variable or database column. The CHARACTER type column is not long enough to hold the result.

System Action Taken: The report was not generated.

Corrective Action: Lengthen the CHAR variable in the DEFINE section of the specification or alter the length of the CHAR column in the database table.

-9063 **Description of Error**: The user function *function-name*, defined by the user in the DEFINE statement of ACE, could not be found in the C function definition table "**userfuncs**" in the user's C static data area.

System Action Taken: The report was not generated.

Corrective Action: Check that the "**userfuncs**" structure in your user-linked ACEGO program contains the required function name. (See examples one and two in the ACEGO and PERFORM section of the INFORMIX-ESQL/C manual.) When linking C functions to an ACE report, the correct command line sequence to run the ACE report

```
newacego
```

where newacego is the user-linked ACEGO program. Do not use the original ACEGO program.

-9064 **Description of Error**: The runtime string table is full. No more memory can be allocated.

System Action Taken: The report was not generated.

Corrective Action: You have exceeded the data space limit on your machine. Reduce the number of unique string and variable names or their length in the report. (Note: identical strings are stored only one time.)

-9065 **Description of Error**: Memory allocation has failed. The user's ACE report specification is too large.

System Action Taken: The report was not generated.

Corrective Action: You have exceeded the data space limit on your machine. Reduce the length or complexity of the ACE report.

-9066 **Description of Error**: An error has occurred while writing to the output report.

System Action Taken: The report was generated but was not saved (operating system error).

Corrective Action: You may have exceeded the available disk storage space. Remove any unneeded files from the disk and try the action again.

-9067 **Description of Error**: ACE's arithmetic stack has exceeded its bounds. The expression here is too complicated.

System Action Taken: The report was not generated.

Corrective Action: Reduce the complexity of the arithmetic expression or divide it into two or more expressions.

-9068 **Description of Error**: An SQL statement has been found in the SELECT section that is not a SELECT statement. Only SELECT statements or SELECT statements that create temporary tables are allowed in the SELECT section.

System Action Taken: The report was not generated.

Corrective Action: Remove any non-SELECT statements (INSERT, UPDATE, etc.) from the SELECT section.

-9140 **Description of Error**: An attempt was made to divide by zero on line *lineno*. The result was set to zero.

System Action Taken: The report was not generated.

Corrective Action: Check the calculations done on the indicated line. If program includes a user variable within an aggregate expression, the variable must be set to a value before the first SELECT statement is read. (This value can be provided through a command line parameter or a PROMPT FOR statement.)

-9142 **Description of Error**: Date conversion error.

System Action Taken: The report was not generated.

Corrective Action: Check any values included in DATE() functions.

-9503 **Description of Error**: The table *table-name* could not be opened.

System Action Taken: The report was not generated (operating system error).

Corrective Action: Check that the SELECT statement does not refer to a table that you do not have operating system permission to read. Contact your System Administrator if you need help with this action. Alternately, you may have exceeded the maximum number of tables allowed (on most systems this is 14 tables). Reduce the number of tables included in your report.

1203 **Description of Error**: Cannot find message file.

System Action Taken: The statement was not processed.

Corrective Action: RDSQL cannot locate a needed message file. Check that both the INFORMIXDIR and DBLANG environment variables are set with the appropriate pathname. Contact your System Administrator if you need help with this action.

1204 **Description of Error**: Type of terminal is unknown to the system.

System Action Taken: The statement was not processed.

Corrective Action: Check the setting of the TERM environment variable. Contact your System Administrator if you need help with this action.

2002 **Description of Error**: You have entered the wrong number of parameters. The syntax is as follows: **formbuild -d** *formname databasename table1 table2 ...* (maximum of 14 tables).

System Action Taken: The form was not generated.

Corrective Action: Check the sequence of parameters present on the command line.

2005 **Description of Error**: Database *databasename* not found or not correct format.

System Action Taken: The requested action was not completed (operating system error).

Corrective Action: Check that the DBPATH environment variable includes the full pathname of the directory holding the database. Contact your System Administrator if you need help with this action.

2008 **Description of Error**: The table *tablename* does not exist in the database.

System Action Taken: The requested action was not completed.

Corrective Action: The table name included in the TABLE section of the form specification file is not found in the database specified in the DATABASE section. Check the spelling of the table name.

2009 **Description of Error**: You have not selected any database tables.

System Action Taken: The requested action was not completed.

Corrective Action: You must include one or more table names in the TABLES section of the form specification file.

2010 **Description of Error**: There is not enough memory to use the default form option for this particular choice of tables.

System Action Taken: The requested action was not completed.

Corrective Action: You have exceeded the data space limits of your machine. Reduce the number of tables included in the form.

2011 **Description of Error**: Default formbuild has run out of memory.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the data space limits of your machine. Reduce the number of tables included in the form.

2012 **Description of Error**: Form output table *formname* could not be opened.

System Action Taken: The compile was not completed.

Corrective Action: You may have exceeded the open file limit of 14 data files (this number includes the output file). Reduce the number of tables included in the form.

2017 **Description of Error**: The default form has exceeded its label limit.

System Action Taken: The compile was not completed.

Corrective Action: A form can use up to 26 one-character fields, 260 two-character fields, and 1000 fields of three characters or more. Reduce the number of tables in the default form.

2018 **Description of Error**: The default form has exceeded its limit of 260 two-character labels.

System Action Taken: The compile was not completed.

Corrective Action: The total number of two-character fields contained in the tables used to generate the default form exceeds the 260 limit. You must delete one or more of the tables containing two-character fields.

2019 **Description of Error**: The default form has exceeded its limit of 26 one-character labels.

System Action Taken: The compile was not completed.

Corrective Action: The total number of one-character fields contained in the tables used to generate the default form exceeds the 26 limit. You must delete one or more of the tables containing one-character fields.

2020 **Description of Error**: The following tables are involved: *table-name.*

System Action Taken: Check accompanying message for indication of system action.

Corrective Action: The indicated tables are involved in the specified error(s).

8008 **Description of Error**: WARNING: User variables used within aggregates may lead to unexpected results. See line *line-number.*

System Action Taken: The compile was not completed.

Corrective Action: Remove the user variable from the aggregate calculation.

8009 **Description of Error**: See error number *errno.*

System Action Taken: Check accompanying message for indication of system action.

Corrective Action: Locate the indicated error message in this appendix.

INFORMIX-TURBO and INFORMIX-MS-NET Error Messages

This section contains the text of error messages that relate primarily to TURBO and the MS-NET versions of **INFORMIX** products. A description of how the system responds to each error and the suggested corrective action is included. These messages are appropriate if you have TURBO or an MS-NET version of an **INFORMIX** product on your system.

-127 **Description of Error**: No primary key.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Run **bcheck** to determine the source of the problem. Repair your table if necessary.

-128 **Description of Error**: No logging.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have attempted to execute a statement that requires logging. Start your database with transactions and reexecute the statement.

-129 **Description of Error**: Too many users.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The shared memory parameter for the maximum number of users was exceeded. Either ask your System Administrator to adjust the parameter, or try the statement again later.

-130 **Description of Error**: Dbspace not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the name of your dbspace. Run the Turbo Monitor to view existing dbspaces.

-131 **Description of Error**: No free disk space.

System Action Taken: The statement containing the error was not processed.

Corrective Action: There is not enough contiguous disk space to complete the operation. Consult your System Administrator for assistance.

-132 **Description of Error**: The row size is too big.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make the fields of the table smaller or create multiple tables with fewer fields.

-133 **Description of Error**: File has audit trail.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot cluster a file with an audit trail. If you want to cluster the file, you must first drop the audit trail.

-134 **Description of Error**: No more locks.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The shared memory parameter for the maximum number of locks was exceeded. Either lock the entire table, wait until more locks are available, or ask your System Administrator to adjust the parameter.

-513 **Description of Error**: Statement not available in TURBO.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Refer to the *Informix-Turbo Programmer's Manual.* Make sure that you are using the appropriate database server (**sqlexec** for non-TURBO, or **sqlturbo** for TURBO).

-514 **Description of Error**: Only DBA can create a table, view or index for another user.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Ask the DBA to perform the action or to grant you the DBA privilege.

-515 **Description of Error**: Statement not available using non-TURBO database server.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Refer to the reference manual for your front-end product. Make sure that you are using the appropriate database server (**sqlexec** for non-TURBO, or **sqlturbo** for TURBO).

-517 **Description of Error**: The size of the index fields is too large or there are too many parts in the index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of fields or the size of the fields in your index.

-518 **Description of Error**: Cannot have extent size equal to zero.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check your CREATE or ALTER TABLE statement for valid extent sizes.

-519 **Description of Error**: Cannot update column to illegal value.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the values in your UPDATE or INSERT statement.

-520 **Description of Error**: Cannot open database partition.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Either the database does not exist or the shared memory parameter for the maximum number of open partitions was exceeded. Use the Turbo Monitor to determine the source of the problem. Ask the System Administrator to adjust the parameter.

-521 **Description of Error**: Cannot lock system catalog *catalog-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: You are not allowed to lock system catalogs. Check the spelling of the table to be locked.

-522 **Description of Error**: Table *table-name* not selected in query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table in the query.

-523 **Description of Error**: You can only recover, repair, or drop a table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check to make sure that the table in the query is not a view.

-524 **Description of Error**: Lock table can only be used within a transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must start logging for the database before executing the statement.

-525 **Description of Error**: Cannot create database - no system permission.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you have write and execute permission on the current directory.

-526 **Description of Error**: Updates are not allowed on a scroll cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not use a scroll cursor with a statement that is declared SELECT FOR UPDATE.

-527 **Description of Error**: Lock mode is not available on this system.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Modify your program to not WAIT FOR LOCK.

-844 **Description of Error**: Statement is too long (maximum 2,048 characters).

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the length of the statement to less than 2,048 characters.

-900 **Description of Error**: Cannot read network user authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the network user authorization file exists on the server, that it is accessible through the network, and that you have read permission.

-901 **Description of Error**: User not found in network user authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Add the user to the network authorization file.

-902 **Description of Error**: User not authorized or too many entries in authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Requires a version configured for more users. Contact Informix Software for assistance.

-904 **Description of Error**: Authorization file not on licensed INFORMIX-SQL server.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reset the INFORMIXDIR variable to the directory that contains the licensed software.

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
Multiuser 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)
subscripting 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