DB2® Server for VSE & VM

Interactive SQL Guide and Reference

Version 6 Release 1

SC09-2674-00

DB2® Server for VSE & VM

Interactive SQL Guide and Reference

Version 6 Release 1

SC09-2674-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

This book is also provided as an online book that can be viewed with the IBM® BookManager® READ and IBM Library Reader™ licensed programs.

First Edition (December 1998)

This edition SC09-2674, applies to Version 6 Release 1, of the IBM DATABASE 2™ Server for VSE & VM Program 5648-A70, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to: IBM Canada Ltd. Laboratory, Information Development, 2G/345/1150/TOR, 1150 Eglinton Ave East, North York, Ontario, Canada M3C 1H7.

You can also send your comments by facsimile to (416) 448-6161 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to **torrcf@ca.ibm.com**; IBMLink™, to **toribm(torrcf)**; IBM/PROFS®, to **torolab4(torrcf)**; IBMMAIL, to **ibmmail(caibmwt9)**; or through our home page at **http://www.software.ibm.com/data/db2/vse-vm/**

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM
ACF/VTAM
BookManager
IBMLink
CICS/VSE
DATABASE 2
DataPropagator
DB2
Distributed Relational Database Architecture
DRDA
IMS
Library Reader
OS/390
QMF
SQL/DS
VM/ESA
VSE/ESA

About This Manual

This manual is a tutorial and reference for IBM DATABASE 2 Server for VSE & VM (DB2 Server for VSE & VM) interactive SQL (ISQL) users in a Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA) or Virtual Machine/Enterprise Systems Architecture (VM/ESA) environment. The manual presents reference information for particular topics, followed by tutorial exercises that illustrate the reference information. Screen images (hereafter referred to as *displays*) that are similar to the displays that you view when performing ISQL exercises are included in the text.

The manual begins with an introduction to:

- The DB2 Server for VSE & VM relational database management system (RDBMS).
- The Structured Query Language (SQL) which manipulates and controls data managed by the RDBMS.
- The Interactive SQL (ISQL) facility, through which SQL statements are issued from a display terminal.

The manual continues with query and management techniques for DB2 Server for VSE & VM tables that are used during an ISQL session. Specific guidelines are provided for the use of ISQL commands, SQL statements, routines, and VM functions.

The examples and exercises that are provided throughout this manual to familiarize you with ISQL are supplemented by answers and example table layouts, both of which are in appendixes.

Note:

- The terms *DB2 Server for VM*, *DB2 Server for VSE*, and *DB2 Server for VSE & VM* are used in this guide to refer to DB2 Server for VM Version 6 Release 1 unless otherwise indicated.
- When the term *CICS* is used in this manual, *CICS/VSE* is implied. The CICS/VSE product is required for DBCS support.
- The term *VSE* refers to VSE/Enterprise Systems Architecture Version 2 Release 1 Modification 2 or later.
- The term *VM* is used in this guide to refer to VM/ESA Version 2 Release 1 or later.

For a quick summary of reference information on ISQL functions, refer to the *DB2 Server for VSE & VM Quick Reference* manual.

How to Use This Manual

The following information provides a brief description of each chapter and appendix in the manual.

The Summary of Changes summarizes the technical and library changes made for Version 6 Release 1 of the database manager product.

Chapter 1, “Getting Started” on page 1, introduces the database manager, SQL, and ISQL facilities, and takes the reader through a typical ISQL session using the display terminal.

Chapter 2, “Querying Tables” on page 21, discusses query techniques for database manager table data. It describes the use of expressions, operators, keywords, predicates, and durations.

Chapter 3, “Manipulating Query Information” on page 41, describes the manipulation of queried information to produce calculated results by using functions, concatenation operators, and constants. A separate section describing how to obtain online reference information is included.

Chapter 4, “Using Query Results” on page 61, discusses the presentation of queried information both on the display and in a printed report.

Chapter 5, “Managing Table Data” on page 71, provides DB2 Server for VSE & VM table management guidelines that include the timing of table changes, referential integrity, and inserting, updating, and deleting table data.

Chapter 6, “Using ISQL Commands to Save Time When Executing Statements” on page 87, explains the use of ISQL commands with SQL statements during an ISQL session.

Chapter 7, “Formatting Query Results” on page 95, offers additional methods of displaying queried information. Formatting techniques for both the display and for printed reports are discussed in detail.

Chapter 8, “Storing SQL Statements” on page 119, describes the use of stored SQL statements, including management and processing methods.

Chapter 9, “Creating and Using Routines” on page 125, discusses the use of database manager routines to process a series of ISQL commands or SQL statements.

Chapter 10, “Using Additional Query Techniques” on page 135, provides additional query techniques, including the use of multiple tables, correlation names, multiple queries, subqueries, views, and unions.

Chapter 11, “Creating and Managing Tables” on page 175, describes procedures to create and maintain a personal set of database manager tables with an emphasis on the use of table keys. In addition, information is provided that explains the sharing of tables with other database manager users. Guidelines that improve query performance are also provided.

DB2 Server for VM

Chapter 12, “Using VM Functions” on page 197, discusses the use of VM commands and functions to enhance the ISQL session, such as the specification of the number of copies of printed reports.

Chapter 13, “ISQL Commands” on page 203, is a reference chapter for ISQL commands that provides a format diagram, a description, and an example, where appropriate, for each command.

Appendix A, “Answers to the Exercises” on page 261, contains the answers to the exercises in the manual.

Appendix B, “Sample Tables” on page 269, describes the structure and contents of the sample database manager tables used in the examples and exercises in the manual.

Appendix C, “Summary of ISQL PF Keys” on page 283, describes the ISQL commands associated with the PF keys on the keyboard.

Appendix D, “Summary of SQL Statements for Interactive Use” on page 285, lists the SQL statements that you can refer to the *DB2 Server for VSE & VM SQL Reference* manual, if you want more information.

DB2 Server for VSE

Appendix E, “Suppressing the ISQL Sign-On Display for DB2 Server for VSE” on page 287, gives you information on suppressing the ISQL sign-on display and related terminal messages.

The Bibliography lists the full titles and order numbers of related publications, and is followed by the Index.

Components of the Relational Database Management System

Figure 1 on page xvi depicts a typical configuration with one database and two users.

Figure 2 on page xvii depicts a typical configuration with one database, one batch partition user, and a CICS® partition with several interactive users.

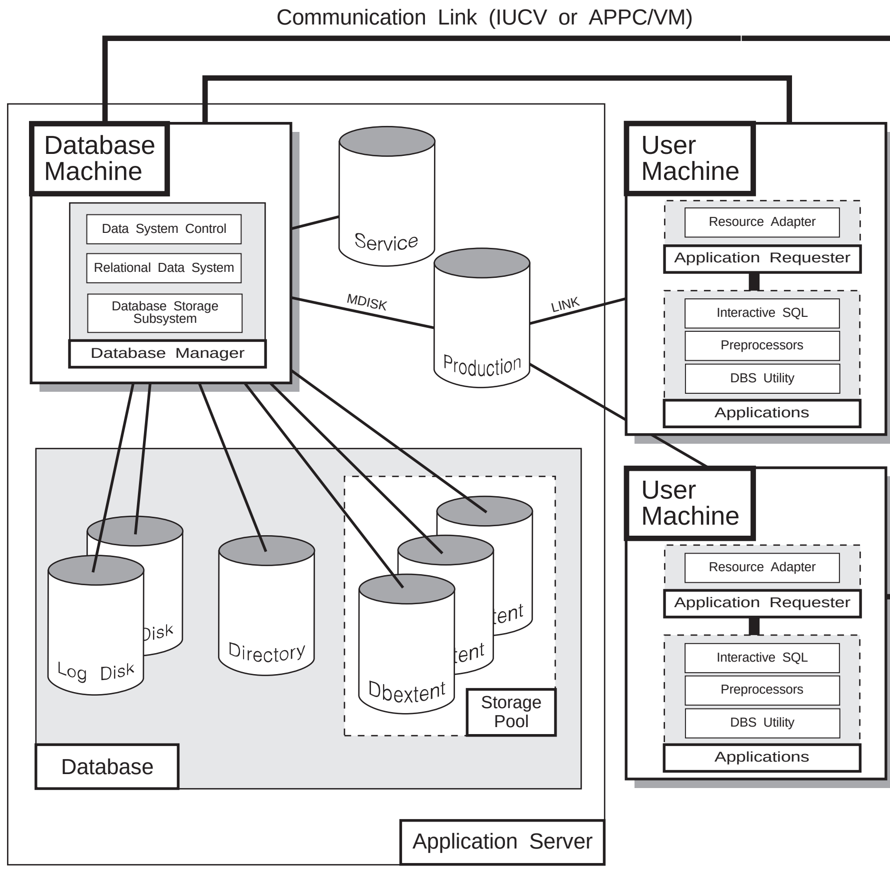


Figure 1. Basic Components of the RDBMS in VM/ESA

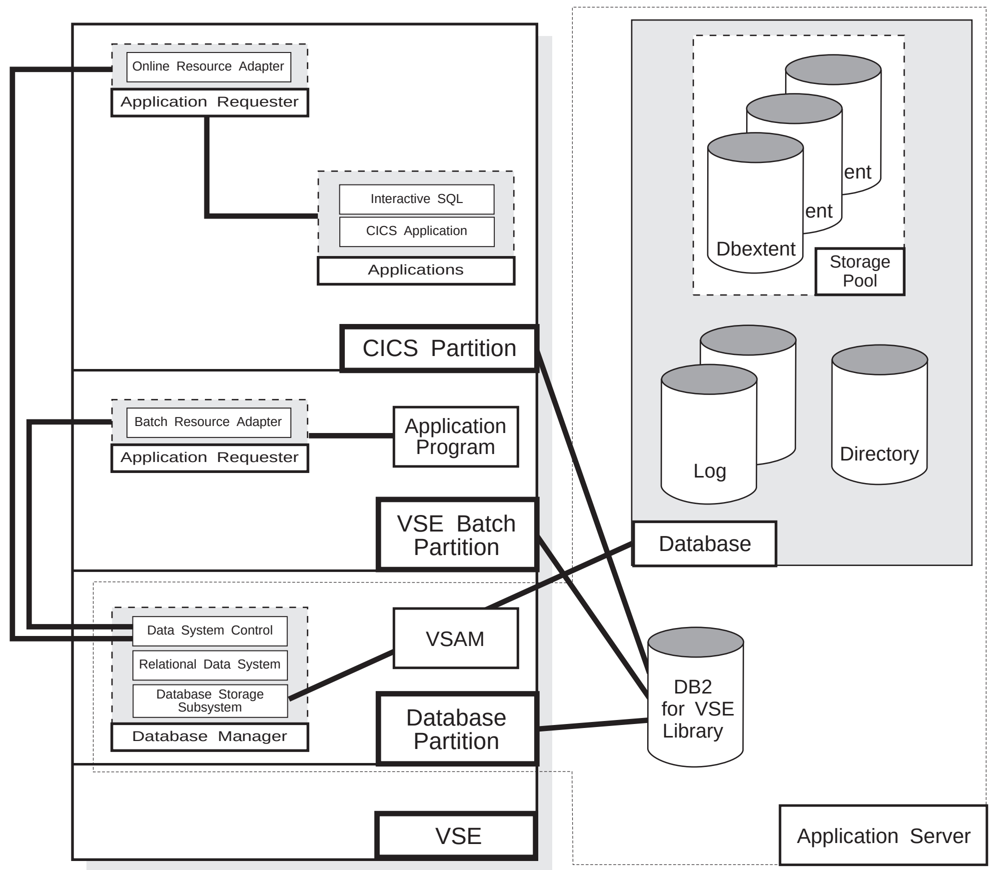


Figure 2. Basic Components of the RDBMS in VSE

The **database** is composed of :

- A collection of data contained in one or more *storage pools*, each of which in turn is composed of one or more *database extents (dbextents).* A dbextent is a VM minidisk.
- A *directory* that identifies data locations in the storage pools. There is only one directory per database.
- A *log* that contains a record of operations performed on the database. A database can have either one or two logs.

The **database manager** is the program that provides access to the data in the database. In VM it is loaded into the database virtual machine from the production disk. In VSE it is loaded into the database partition from the DB2 Server for VM library.

The **application server** is the facility that responds to requests for information from and updates to the database. It is composed of the database and the database manager.

The **application requester** is the facility that transforms a request from an application into a form suitable for communication with an application server.

Prerequisite Publications

Although not required, you should have an understanding of material covered in the *DB2 Server for VSE & VM Overview* manual.

Corequisite Publications

The following manuals should be used with this manual:

DB2 Server for VSE & VM SQL Reference
DB2 Server for VM Database Administration
DB2 Server for VSE Database Administration
DB2 Server for VSE & VM Overview.

Syntax Notation Conventions

Throughout this manual, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right and from top to bottom, following the path of the line.

  The >>─── symbol indicates the beginning of a statement or command.

  The ───> symbol indicates that the statement syntax is continued on the next line.

  The >─── symbol indicates that a statement is continued from the previous line.

  The ───>< symbol indicates the end of a statement.

  Diagrams of syntactical units that are not complete statements start with the >─── symbol and end with the ───> symbol.

- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:

```
►►──SAVE──────────────────────────────────────────────────────►◄
```

  Others must be followed by one or more keywords or variables. For example:

```
►►──SET AUTOCOMMIT OFF────────────────────────────────────────►◄
```

- Keywords may have parameters associated with them which represent user-supplied names or values. These names or values can be specified as either constants or as user-defined variables called *host_variables* (*host_variables* can only be used in programs).

```
►►──DROP SYNONYM──synonym─────────────────────────────────────►◄
```

- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARacter). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses, or arithmetic operators are shown, you must use them as indicated by the syntax diagram.
- All items (parameters and keywords) must be separated by one or more blanks.
- Required items appear on the same horizontal line (the main path). For example, the parameter *integer* is a required item in the following command:

```
►►──SHOW DBSPACE──integer──────────────────────────────────────────►◄
```

  This command might appear as:

  ```
  SHOW DBSPACE 1
  ```

- Optional items appear below the main path. For example:

```
►►──CREATE──┬────────┬──INDEX──────────────────────────────────────►◄
            └─UNIQUE─┘
```

  This statement could appear as either:

  ```
  CREATE INDEX
  ```

  or

  ```
  CREATE UNIQUE INDEX
  ```

- If you can choose from two or more items, they appear vertically in a stack.

  If you must choose one of the items, one item appears on the main path. For example:

```
►►──SHOW LOCK DBSPACE──┬─ALL─────┬─────────────────────────────────►◄
                       └─integer─┘
```

  Here, the command could be either:

  ```
  SHOW LOCK DBSPACE ALL
  ```

  or

  ```
  SHOW LOCK DBSPACE 1
  ```

  If choosing one of the items is optional, the entire stack appears below the main path. For example:

```
►►──BACKWARD──┬───────────┬──────────────────────────────────────────►◄
              ├─integer───┤
              └─MAX───────┘
```

Here, the command could be:

```
BACKWARD
```

or

```
BACKWARD 2
```

or

```
BACKWARD MAX
```

- The repeat symbol indicates that an item can be repeated. For example:

```
             ┌──────┐
►►──ERASE────▼─name─┴───────────────────────────────────────────────►◄
```

This statement could appear as:

```
ERASE NAME1
```

or

```
ERASE NAME1 NAME2
```

A repeat symbol above a stack indicates that you can make more than one choice from the stacked items, or repeat a choice. For example:

```
                   ┌─,─────────────────────┐
►►──VALUES──(──────▼─┬─constant──────────┬─┴──)──────────────────────►◄
                     ├─host_variable_list┤
                     ├─NULL──────────────┤
                     └─special_register──┘
```

- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.

```
      ┌─ASC──┐
►►────┼──────┼───────────────────────────────────────────────────────►◄
      └─DESC─┘
```

- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

elIn the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you do not enter PCTFREE =, the database manager will set it to the default value of 10.

```
      ┌PCTFREE = 10──────┐
►►────┤                  ├──────────────────────────────────────────────►◄
      └PCTFREE = integer─┘
```

- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:

```
                  ┌PRIVILEGES┐
►►──REVOKE ALL────┴──────────┴──────────────────────────────────────────►◄
```

 Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

- Sometimes a single parameter represents a fragment of syntax that is expanded below. In the following example, **fieldproc_block** is such a fragment and it is expanded following the syntax diagram containing it.

```
►►────┬──────────────────────────────┬──┤ fieldproc_block ├────────────►◄
      └NOT NULL──┬────────────────┬──┘
                 ├─UNIQUE─────────┤
                 └─PRIMARY KEY────┘
```

```
fieldproc_block:
├──FIELDPROC──program_name──┬─────────────────────────┬──────────────────┤
                            │      ┌─,─────────┐      │
                            └─(────▼─constant──┴───)──┘
```

SQL Reserved Words

The following words are reserved in the SQL language. They cannot be used in SQL statements except for their defined meaning in the SQL syntax or as host variables, preceded by a colon.

In particular, they cannot be used as names for tables, indexes, columns, views, or dbspaces unless they are enclosed in double quotation marks (").

ACQUIRE
ADD
ALL
ALTER
AND
ANY
AS
ASC
AVG

BETWEEN
BY

CHAR
CHARACTER
COLUMN
COMMENT
COMMIT
CONCAT
CONNECT
COUNT
CREATE
CURRENT

DBA
DBSPACE
DELETE
DESC
DISTINCT
DOUBLE
DROP

EXCLUSIVE
EXECUTE
EXISTS
EXPLAIN

FIELDPROC
FOR
FROM

GRANT
GRAPHIC
GROUP

HAVING

IDENTIFIED
IN
INDEX
INSERT
INTO
IS

LIKE
LOCK
LONG

MAX
MIN
MODE

NAMED
NHEADER
NOT
NULL

OF
ON
OPTION
OR
ORDER

PACKAGE
PAGE
PAGES
PCTFREE
PCTINDEX
PRIVATE
PRIVILEGES
PROGRAM
PUBLIC

RESOURCE
REVOKE
ROLLBACK
ROW
RUN

SCHEDULE
SELECT
SET
SHARE
SOME
STATISTICS
STORPOOL
SUM
SYNONYM

TABLE
TO

UNION
UNIQUE
UPDATE
USER

VALUES
VIEW

WHERE
WITH
WORK

Authorization Names and Passwords

Authorization names and passwords are limited to 8 characters and cannot have embedded blanks.

Authorization Names without Quotation Marks

The name *must* begin with a letter, $, #, or @ and contain letters, numbers, $, #, @, or underscore. Avoid using SQL reserved words and Database Services Utility reserved words. See the *DB2 Server for VSE & VM Quick Reference* manual for a list of these reserved words.

DB2 Server for VM

Note: # is the usual CP TERMINAL LINEND character.

Authorization Names in Double Quotation Marks

Names can begin with any character and contain any combination of characters when enclosed in double quotation marks. However, the double quotation mark character itself is not allowed within the names, and leading blanks cause errors.

Use of Highlighting in This Guide

Database manager commands and statements are illustrated throughout this manual using

```
indented and highlighted type.
```

You can type these commands and statements. Commands or statements that are

```
indented but not highlighted
```

illustrate additional examples and options. If you type them, you may produce results different from those shown in this manual.

Titles of publications, command variables, parameter values, character strings, and the first use of a term are printed in *italics*.

Any information appearing on the display that is referred to in the text is highlighted in this manual. For example, if the term User ID appears on the display, an instruction to the reader to make an entry beside the term is written in the manual as: Type your user ID in the `User ID` input area. Note the special highlighting for User ID.

Uppercase characters are used for:

- Acronyms (for example, DB2 Server for VSE & VM)
- ISQL commands, statements, and instructions (for example, the CHANGE command)
- Names on top of keys (for example, PF3)
- Names of programs, macros, and EXECs (for example, the PROFILE EXEC)
- Option names, keywords, and special registers (for example, the CASE keyword of the SET command)
- Datasets and files, including tables (for example, the ACTIVITY table).

Italics emphasize the importance of the *italicized phrase*.

Summary of Changes for DB2 Version 6 Release 1

This is a summary of the technical changes to the DB2 Server for VSE & VM Version 6 Release 1 database management system. All manuals are affected by some or all of the changes discussed here. This summary does not list incompatibilities between releases of the DB2 Server for VSE & VM product; see either the *DB2 Server for VSE & VM SQL Reference*, *DB2 Server for VM System Administration*, or the *DB2 Server for VSE System Administration* manuals for a discussion of incompatibilities. Version 6 Release 1 of the DB2 Server for VSE & VM database management system is intended to run on the Virtual Machine/Enterprise Systems Architecture (VM/ESA®) Version 2 Release 2 or later environment and on the Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA™) Version 2 Release 2 or later environment.

Enhancements, New Functions, and New Capabilities

DRDA® RUOW Application Requestor for VSE (Online)

DRDA Remote Unit of Work Application Requestor provides read and update capability in one location in a single unit of work.

This support provides CICS/VSE® online application programs with the ability to execute SQL statements to access and manipulate data managed by any remote application server that implements the DRDA architecture. Online application programs that access remote application servers need to be preprocessed to create a bind file and then bound (using CBND) to the remote application server. Online application programs that access a local application server are preprocessed as in previous releases.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Application Programming*
- *DB2 Server for VSE Installation*

Stored Procedures

The ability to use stored procedures provides distributed solutions that let more people access data faster.

A stored procedure is a user-written application program compiled and stored at the server. When the database is running in multiple user mode, local applications or remote DRDA applications can invoke the stored procedure. SQL statements are local to the server and issued by a stored procedure so they do not incur the high network costs of distributed statements. Instead, a single network send and receive operation is used to invoke a series of SQL statements contained in a stored procedure.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VM System Administration*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM Operation*

TCP/IP Support for DB2 Server for VM

TCP/IP support allows:

- VM applications to use SQLDS-private protocol to connect to VM databases over TCP/IP.
- VM applications to use DRDA protocol to connect to DB2 family databases (and any other database that supports DRDA connections) over TCP/IP.
- non-VM applications to use DRDA-protocol to access VM database over TCP/IP.

TCP/IP support for DB2 Server for VM integrated with the DB2 Server for VM application server means a system easier to configure and maintain.

The database manager will optionally secure TCP/IP connections using any external security manager that supports the RACROUTE interface.

New Code Page and Euro Symbol Code Page Support

The following CCSIDs are now supported:

- 1112: Latvian/Lithuanian
- 1122: Estonian
- 1123: Ukrainian
- 1130: Vietnamese
- 1132: Lao
- 1148: E-International
- 1140: E-English
- 1141: E-German
- 1144: E-Italian
- 1147: E-French

Additional support has been added for conversions from Unicode (UCS-2) to host CCSIDs.

For a complete list of CCSIDs supported refer to the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals.

DataPropagator™ Capture

DataPropagator Capture is part of the DB2 Family of DataPropagator products. DataPropagator Capture is updated for Version 6 Release 1 compatibility.

QMF for VM, QMF for VSE, and QMF for Windows®

IBM Query Management Facility (QMF™) is now an separately priced feature of DB2 Server for VSE & VM. QMF is a tightly integrated, powerful, and reliable tool that performs query and reporting for IBM's DB2 relational database Management System Family. It offers an easy-to-learn, interactive interface. Users with little or no data processing experience can easily retrieve, create, update, insert, or delete data that is stored in DB2.

QMF offers a total solution that includes accessing large amounts of data and sharing central repositories of queries and enterprise reports. It also allows you to implement tightly-controlled, distributed, or client-server solutions. In addition, you can use QMF to publish reports to the World Wide Web that you can view with your favorite web browser.

Using QMF, users can access a wide variety of data sources, including operational or warehouse data from many platforms: DB2 for VSE, VM, OS/390® and Windows. Via IBM Data Joiner, you can access non-relational data, such as IMS™ and VSAM, as well as data from other vendor platforms.

RDS Above the Line

The RDS component will load and execute above the 16 megabyte line. This support frees up approximately 1.5 megabytes of storage below the 16 megabyte line (or approximately 2.5 megabytes, if DRDA is installed) when compared to Version 5 Release 1. No installation or migration changes are required for this support to be used (except for the definition of VM Shared Segments and for users who execute the database server with AMODE(24)). If sufficient storage is available, the RDS component will be automatically loaded above the 16 megabyte line. When using VM Shared Segments, the RDS Segment should be defined above the 16 megabyte line.

VM users who wish to run the database server in 24-bit addressing mode (i.e. use the AMODE(24) parameter) **must** use a virtual storage size no greater than 16 megabytes. See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* for release to release incompatibility information.

Combining of NLS Feature Installation Tapes with Base Product Installation Tape

All available NLS features for DB2 Server for VSE, DB2 Server for VM, Control Center for VSE and REXX SQL for VM have been combined with the respective base product installation tape. Customers interested in an NLS feature language will no longer need to order an additional NLS feature tape because all NLS languages will be available to all customers. In all cases, the default language as shipped is American English. The installation and migration processes have been changed to allow you to choose the default language. Refer to the *DB2 Server for VM Program Directory*, *DB2 Server for VSE Installation*, *DB2 for VSE Control Center Installation and Operations Guide*, and *DB2 REXX SQL for VM/ESA Installation* for the details of how these changes affect the installation process and how you can choose to have a different default language.

Control Center Feature

DB2 Server for VSE & VM Version 6 Release 1 enhances the new Control Center feature as follows:

For both VM/ESA and VSE/ESA:

- Access to the Query Management Facility (QMF)

For VM/ESA:

- Compatibility with DB2 Server for VM Version 6 Release 1 initialization parameters and operator commands
- Shared File System Support (SFS) in a VM/ESA environment
- CA-DYNAM/T Interface Support in a VM/ESA environment
- Data Restore Incremental Backup Support in a VM/ESA environment

For VSE/ESA:

- Control Center code installation on any library
- Ability to use while viewing a list of tables online
- Ability to create, reorganize, unload, reload, move and copy tables in batch mode
- Ability to update table statistics in batch mode
- Ability to drop tables online

Data Restore Feature

The Data Restore feature provides archiving and recovery functions in addition to those provided in DB2 for VSE & VM. Data Restore is enhanced in Version 6 Release 1 with incremental database archiving support. The support allows you to archive only the areas of the database that have been updated since the last database archive, instead of having to archive the entire database. This can provide significant savings for customers with large databases which are updated infrequently, or where only a small fraction of the database is updated frequently.

DB2 REXX SQL Feature

The DB2 REXX SQL feature provides a REXX interface for VM customers to allow SQL calls to be executed from REXX programs. The DB2 REXX SQL feature is updated for Version 6 Release 1 compatibility.

Reliability, Availability, and Serviceability Improvements

Migration Considerations

Migration is supported from SQL/DS™ Version 3 and DB2 Server for VSE & VM Version 5. Migration from SQL/DS Version 2 Release 2 or earlier releases is not supported. Refer to the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual for migration considerations.

Library Enhancements

Some general library enhancements include:

- The following books have been removed from the library:
  - *DB2 Server for VM Operation*
  - *DB2 Server for VSE Operation*
  - *DB2 Server for VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE Interactive SQL Guide and Reference*
  - *DB2 Server for VM Database Services Utility*
  - *DB2 Server for VSE Database Services Utility*
- The following books have been added to the library:
  - *DB2 Server for VSE & VM Operation*
  - *DB2 Server for VSE & VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE & VM Database Services Utility*

Refer to the new *DB2 Server for VSE & VM Overview* for a better understanding of the benefits DB2 Server for VSE & VM can provide.

Chapter 1. Getting Started

This chapter introduces the database manager and the Interactive SQL (ISQL) facility, and shows how to access both.

With the ISQL facility, you can manipulate data contained in a relational database from a display terminal. In the ISQL environment, you will learn such procedures as controlling the display, interpreting database manager messages, entering ISQL commands, and stopping ISQL.

Introducing the DB2 Server for VSE & VM Database Manager

The DB2 Server for VSE & VM relational database management system uses the Structured Query Language (SQL) to manage stored data.

Using SQL, you can query, add, delete, and update data. The language consists of a collection of statements, each of which performs a particular function.

This manual describes how to use the database manager interactively from a CICS display terminal. The terminals supported are IBM 3277, 3278, 3279, or 3290 (or equivalent) with a line length of at least 80 characters and at least 24 lines per display. The database manager also supports the larger display sizes offered by some models of the 3278 and 3279 terminals.

Designed for the interactive user, the manual gives examples of those functions and statements that can be used interactively. For a more comprehensive description of database manager functions, as well as SQL statements used for querying and displaying data, see the *DB2 Server for VSE & VM SQL Reference* manual.

All data stored in the database is in the form of *tables*. The person who creates the table also names it. The table shown in Figure 3 is named `CARS`.

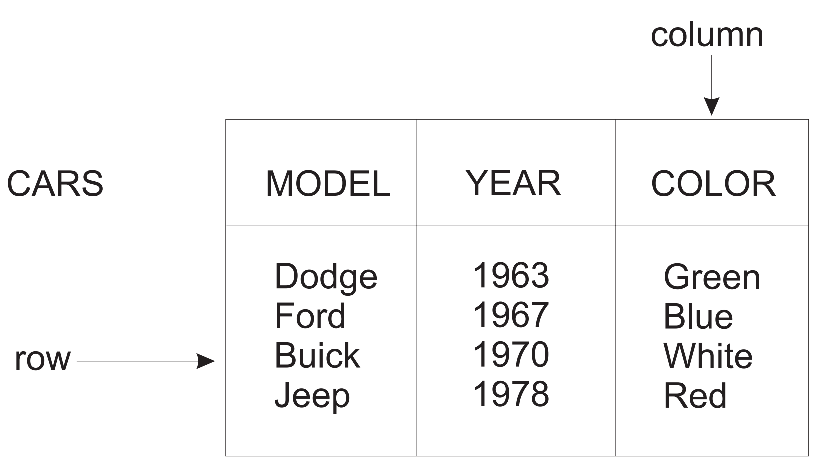


Figure 3. A DB2 Server for VSE & VM Table

A table consists of (vertical) *columns* and (horizontal) *rows*. Each column has a name; the columns in the `CARS` table are `MODEL`, `YEAR`, and `COLOR`.

A *value* is found at the intersection of a column and a row; for example, in the third row of the CARS table, the information in the COLOR column is the value White.

You usually require several tables to adequately store information for an organization. To illustrate how information is stored and used, a set of sample tables is provided for your use. These tables reside in a sample *relational database*, For DB2 Server for VM users, who request access to this relational database are generally granted their own individual online copies. For DB2 Server for VSE users, if your administrator used the IBM-supplied routine ARINEWUS to set you up as a new ISQL user, you receive a copy of the sample tables with all privileges on these tables. Your copy ensures that the table data will remain uncorrupted by other users, which sometimes occurs when multiple users have access to the same data.

If you do not have a copy of the sample tables, you can still do the exercises, but you must use the prefix SQLDBA. with the table names. The sample tables are described and illustrated in Appendix B, “Sample Tables” on page 269, and are referred to throughout this manual.

This book uses simple examples and samples, but this database manager can readily be used for complex applications in many environments, including scientific, technological, and academic.

Introducing ISQL

SQL statements retrieve, add, delete, and update data in tables, and can be either embedded or interactive. The *Embedded SQL* statements are coded within an application program, and do not begin until the program is being run. SQL statements that are issued interactively, by comparison, create an immediate program or system response for each statement the user issues at a display terminal. This is *interactive processing*, and it is the focus of this manual.

You can issue statements or commands from a display terminal through the interactive SQL (ISQL) facility. Using the following ISQL commands, you can work with the database manager from a display terminal to:

- Control the Display of Data

  You can control the display of data that results from a query in several ways. For example, you can scroll through the results of a query that has more rows than can fit on one display, or look at results that are too wide for the display.

- Print Reports

  You can create reports that are based on data in tables. You can also modify these reports to fit your needs with titles, page numbers, dates and totals.

- Enter Data

  You can enter one or more rows of data into an existing table with the ISQL INPUT command.

- Obtain Online HELP Information from a Display Terminal

  If online HELP information was loaded during installation, you can obtain reference information on your display for ISQL topics. The topics available include reference information about SQL statements, ISQL commands, and messages.

Online HELP information may also have been installed on your system in other national languages. If you want HELP information in one of these languages, you can specify the language for online HELP by using the SET LANGUAGE command.

- Store SQL Statements for Repetitive Use

  You can store SQL statements that are used frequently. A name is assigned to each stored statement to identify it for future use.

- List Operating Characteristics

  You can inquire about operating characteristics that are set using the ISQL SET command. For example, you can see the character that is displayed in null fields.

- Use Routines

  You can store routines, which consist of a series of ISQL commands, SQL statements, or both, and run them at a later time. A routine is especially useful for a frequently used sequence of commands and statements. Routines are discussed in detail in Chapter 9, “Creating and Using Routines” on page 125.

- Switch between Application Servers

  With the CONNECT statement, you can access other application servers. You can access any application servers that have implemented the DRDA protocol. For more information about the CONNECT statement, see the *DB2 Server for VSE & VM SQL Reference* manual.

Introducing the ISQL Display Terminal

ISQL can be run on a variety of display terminals, including the larger display sizes offered by some models of the IBM 3278 and 3279 (or equivalent) display devices. ISQL also supports 5550 terminals with double-byte character sets.

The amount of data displayed varies according to the dimensions of the display terminal being used. Examples in this book are usually a 24-line by 80-character display.

DB2 Server for VSE

Note: The 62 x 160 display requires a CICS/VSE terminal. In an SNA environment, it requires CICS 1.6 ACF/VTAM® Release 1 or later or TCAM Version 2 Release 3 or later.

Using the Program Function Keys

Most keyboards have a group of special keys called program function (or just PF) keys. You use them for quick entry of common ISQL commands. Use of the keys is explained as you proceed through this manual. A summary of the PF keys is in Appendix C, “Summary of ISQL PF Keys” on page 283.

If your PF keys do not match those described in the summary, you can change their functions and tailor them to your needs. Consult the appropriate person in your organization to determine the customized key setting.

Defining the ISQL Session

An ISQL session is signing on, starting ISQL, performing a task (or tasks), and then stopping ISQL.

Before using ISQL from a display terminal, consult the appropriate person in your organization to obtain the following:

DB2 Server for VSE

- A *user ID*. This is a unique user identification that identifies you to the database manager. The user ID and password are optional. If you do not want to type a user ID or password, press ENTER, and the default user ID and password are used. This lets you perform certain activities as defined by your site.
- A *password*. This identification is yours exclusively, and should be kept secret. The user ID and password are optional. If you do not want to type a user ID or password, press ENTER, and the default user ID and password are used. This lets you perform certain activities as defined by your site.
- Access to the sample tables. A description of how your database administrator (DBA) can provide these tables for you is in the *DB2 Server for VSE Database Administration* manual. The tables must be created as described in the *DB2 Server for VSE Database Administration* manual to ensure that the examples and exercises produce the results described in this manual.

DB2 Server for VM

- Access to ISQL. You must have a user ID (to identify you to the VM system) and a password. You must also have authorization to connect with the database manager. This authorization is generally granted by someone with *database administrator* (*DBA*) authority.
- Access to the sample tables. A description of how your DBA can provide these tables for you is in the *DB2 Server for VM Database Administration* manual. The tables must be created as described in the *DB2 Server for VM Database Administration* manual to ensure that the examples and exercises produce the results described in this manual.

Note: Your site may have a different signon procedure than that shown on the following pages. Consult the appropriate person in your organization for the correct procedure.

To access ISQL in the VM system environment, you must do the following:

1. Log on the VM system.
2. Start IPL to load the Conversational Monitor System (CMS).
3. Start ISQL.

Each of the activities is described later in this chapter.

Using DBCS for DB2 Server for VM

Some languages, such as Japanese and Korean, require double-byte character sets. If you want to input or see double-byte character sets (DBCS) during your ISQL session, you must enter the following CMS command before starting ISQL:

```
SET FULLSCREEN ON
```

The SET FULLSCREEN ON command lets you input DBCS characters in CMS command mode, and allows input and display of DBCS characters in ISQL command and display mode.

If you are not going to work with DBCS, you do not need this command.

Before Starting ISQL

DB2 Server for VSE

The online resource adapter (ORA) must be enabled before you can start ISQL. ISQL accesses the application server to which the ORA is connected. The ORA connects multiple application servers at a time. You specify the application servers by the DBNAME parameter of the CIRB transaction.

After the online resource adapter is started, you can use the CIRA transaction to add connections to other DB2 Server for VSE & VM application servers. CIRA can be entered multiple times with different *server_name*s to establish connections to the specified application server. With one CIRA command, you can also establish a list of *server_name*s. The system operator or database administrator (DBA) usually performs these tasks.

While in ISQL you can enter a null CONNECT statement to display the connected user ID and application server names.

DB2 Server for VM

Before you can start ISQL, the following steps must be completed for you:

- The DB2 Server for VM disks must be linked.
- The application server must be started.
- The SQLINIT EXEC must be run.

The first two tasks are usually performed by the system operator or the database administrator (DBA), and the SQLINIT EXEC is usually automatically run when you log on your user ID.

The SQLINIT EXEC establishes the required links and defines the name of the application server. If the SQLINIT EXEC is not automatically run for you, run the EXEC before you start ISQL. You must know the name of the application server. If you do not know the name of the application server, speak to your DBA. In the following example, to run the SQLINIT EXEC establishing a link to the SAMPLEDB server, type the following and press ENTER:

```
sqlinit dbname(sampledb)
```

If you want to use the SQLINIT EXEC, refer to the *DB2 Server for VSE Database Administration* and *DB2 Server for VM Database Administration* manuals.

Starting ISQL for DB2 Server for VSE

ISQL runs as a CICS/VSE transaction. A CICS user invokes this transaction just like any other CICS transaction.

After CICS has been activated and the DB2 Server for VSE & VM online support has been started, the CICS user starts ISQL by entering the following four-character CICS transaction identifier from a CICS terminal, and pressing ENTER:

isql

ISQL responds with a display like the one shown in Figure 4 on page 7.

The screen displays the default application server to which the ORA is connected.

The target application server can be changed by entering the target database information as in the following example:

```
Enter User ID, Password and Target Database, then press Enter

    User ID ==========> ________
    Password =========> ________
    Target Database ==> SQLDB1_TOR_INV
```

This is also the application server to which ISQL will be connected to subsequently. At this point, because ISQL is not connected to the application server to which the ORA is connected, you can end the ORA and restart to another database. If you log on to ISQL again, the signon screen is redisplayed showing the target application server to which the ORA is now connected.

When not connected to a target application server, the ISQL system displays the line `Online Support is not ready. Please exit ISQL`. The online resource adapter (ORA) must be enabled before you can start ISQL. For additional information on enabling the ORA, and other requirements before you can start ISQL, refer to "Before Starting ISQL" on page 5 and the *DB2 Server for VSE System Administration* and *DB2 Server for VM System Administration* manuals.

Alternative Methods for Starting ISQL

If you decide to press ENTER instead of specifying your user ID and password when the signon display is displayed, then you must use the explicit database manager CONNECT statement as follows:

CONNECT *authorization_name* IDENTIFIED BY *password* TO *server_name*

An exception to using the CONNECT statement in the above situation is if your installation has defined a default *authorization_name* for you, in which case, you do not have to specify your *authorization_name*, password, nor *server_name*.

If the TO parameter is not specified, then the connection to the previously connected server will be maintained.

An alternate method for invoking ISQL without having to use the ISQL signon display is described in Appendix E, "Suppressing the ISQL Sign-On Display for DB2 Server for VSE" on page 287.

```
    Welcome to the interactive SQL facility of DB2 for VSE


             IIIIIIII  SSSSSSSS  QQQQQQQQQ   LL
                II     SS        QQ     QQ   LL
                II     SSSSSSSS  QQ     QQ   LL
                II           SS  QQ   QQ QQ  LL
             IIIIIIII  SSSSSSSS  QQQQQQQQQ   LLLLLLLL
                                         QQ

          Default Target Database is SQLDS

  Enter User ID, Password and Target Database, then press Enter

      User ID ==========>
      Password =========>
      Target Database ==>


To exit now, enter EXIT in user ID field with no password, press Enter.
  To exit later, use the EXIT command. Use the HELP command for help.
```

Figure 4. Initial ISQL Screen

You may run an ISQL routine as part of the ISQL signon procedure. Refer to "Using the ISQL Transaction Identifier (DB2 Server for VSE)" on page 126 and Appendix E, "Suppressing the ISQL Sign-On Display for DB2 Server for VSE" on page 287.

Signing On by Using the Signon Display

Signon is accomplished by:

1. Entering your user ID at the location identified by the cursor (two positions to the right of `User ID ==>`). There must be one, and only one, blank between the > and your user ID.
2. Positioning the cursor two positions to the right of `Password ==>` and typing your DB2 Server for VSE & VM password. There must be one, and only one, blank between the > and your password. (You can use the tab key to position the cursor to the correct position. Tab is the key with the arrow pointing to a vertical line on the right side of the key.)

   The area to the right of `Password ==>` is a dark field; characters typed in this area remain invisible.
3. Entering the target application server identified by the cursor (two positions to the right of `Target Database ==>`). There must be one, and only one, blank between the > and the target application server.
4. Pressing ENTER.

Note: On some occasions, your display may lock up and you are unable to type data. If this happens, simply press RESET, ensure that the cursor is in the correct position, and retype the information.

When ISQL recognizes your signon name and password, it responds with the display shown in Figure 5 on page 8.

```
ARI7399I The ISQL default profile values are in effect.
ARI7079I ISQL initialization complete.
ARI7080A Please enter an ISQL or SQL command.



_
                                              Enter a command
```

Figure 5. Initial ISQL Display

Leaving ISQL from the Signon Display

You can exit ISQL from the signon display by typing `EXIT` in the user ID field, leaving the password field blank, and pressing ENTER. If you enter `EXIT` in the user ID field, but also enter a password, ISQL will treat it as a user ID and continue processing.

If you enter `EXIT` to end ISQL, message `ARI7601I` is displayed as follows:

```
ARI7601I ISQL ended normally by your request.
```

Starting ISQL for DB2 Server for VM

Now type ISQL (or the name of your EXEC for ISQL) as follows:

```
isql
```

Press ENTER.

ISQL responds with a display similar to the one shown in Figure 6 on page 9.

```
Ready; T=0.01/0.03 13:41:49
isql
ARI0659I Line-edit symbols reset:
         LINEND=# LINEDEL=OFF CHARDEL=OFF
         ESCAPE=OFF TABCHAR=OFF
ARI0662I EMSG function value reset to ON.
ARI0320I The default server name is SAMPLEDB.
ARI7716I User SQLUSER1 connected to server SAMPLEDB.
ARI7399I The ISQL default profile values are in effect.
ARI7079I ISQL initialization complete.
ARI7080A Please enter an ISQL command or an SQL statement
```

Figure 6. Initial ISQL Display

To use SQL/DS™ Version 1 input and output facilities, specify V1SCRIO in the ISQL EXEC. For example, instead of typing **isql**, type:

isql v1scrio

Press ENTER.

If you specify V1SCRIO, ISQL uses the input and output display commands that were used before the VM/SP 5 version. These older commands support the terminal characters (such as Line-end and Escape) and the RETRIEVE command that are available in SQL/DS Version 1.

You cannot use the V1SCRIO option under the following conditions:

- When you are working with a *double-byte character set* (DBCS). If you invoke ISQL with V1SCRIO in FULLSCREEN MODE, ISQL performs a SET FULLSCREEN OFF at the outset.
- When you are working with large logical screens, such as a model 3290, which is 160 characters by 62 lines, or a model 8514, which is 128 characters by 53 lines. ISQL calculates the logical screen area and does not use the V1SCRIO option if the calculation is greater than 3960 bytes.

Other optional parameters that you can use with the ISQL EXEC, including the V1SCRIO and ROUTINE parameters, are described in "Routines to Which Parameters Can Be Passed (DB2 Server for VM)" on page 125.

Controlling the Display

If your system default is not set for the full-screen environment, you can set it by typing SET FULLSCREEN ON on a CMS command line. Full-screen CMS uses several preset PF keys and displays the current PF key settings in the bottom portion of the display. Before you start ISQL, the PF settings reflect those for the CMS environment. For example, keying PF12 lets you type a command on the command line. After you start ISQL, the PF key settings displayed change to the ISQL settings, and the display is similar to the one shown in Figure 7 on page 10.

```
                                Fullscreen CMS         Columns 1 - 79 of 81

Ready; T=0.01/0.03 13:41:49
isql
ARI0659I Line-edit symbols reset:
         LINEND=# LINEDEL=OFF CHARDEL=OFF
         ESCAPE=OFF TABCHAR=OFF
ARI0662I EMSG function value reset to ON.
ARI0320I The default server name is SAMPLEDB.
ARI7716I User SQLUSER1 connected to server SAMPLEDB.
ARI7399I The ISQL default profile values are in effect.
ARI7079I ISQL initialization complete.
ARI7080A Please enter an ISQL command or an SQL statement








PF1=HELP      2=START     3=          4=           5=RECALL     6=
PF7=          8=          9=HOLD     10=          11=          12=RETRIEVE
```

Figure 7. Initial ISQL Display with Full-Screen CMS

All commands you type appear in the input area near the bottom of the display. All data returned by ISQL appears on a different display; when you exit from such a display, a display similar to the one shown in Figure 7 is returned.

If you want to temporarily suspend the full-screen option, see the *VM/ESA: CMS Command Reference* manual, for information on the SET FULLSCREEN command.

Interpreting DB2 Server for VSE & VM Messages

The system displays messages about certain operating conditions for your terminal session. (See Figure 6 on page 9 for examples of the system messages.) You can receive messages in the language you want, depending on your site. The text of the messages that you receive may be slightly different from those shown here.

Messages have two parts. The first part is the message number, which remains the same regardless of the language setting. An example of a message number is `ARI0503E`. It starts with the letters `ARI`, which identify it as a DB2 Server for VSE & VM message. Then it contains a four-digit number to identify the message. Finally, it ends with one of the following letters that indicates the message type:

I An informational message is displayed.
W A system wait message is displayed.
E An error has occurred and may require some action on your part.
A An action on your part is required.
D Your decision and reply is required.

The second part is the text of the message; for example, `An SQL error has occurred`. ISQL uses the same language for messages as your CMS language setting. For more information, see “SET” on page 245. In most situations, the text is self-explanatory. If it is not, you can use the message number with the HELP command. (For information about using the HELP command to display the message description, refer to Chapter 13, “ISQL Commands” on page 203.) You can also use the message number to look up the message description in the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals.

Messages ARI0503E, ARI0505I, and ARI0504I are usually encountered when an error is detected while the system is processing an SQL statement. Message ARI0504I is an informational message that provides data useful to those who are responsible for locating problems within the system. You can usually ignore this message, but there may be occasions when you are prompted to record its contents.

Message ARI0504I always follows message ARI0505I. Message ARI0503E indicates that the SQL statement being processed was unsuccessful. Message ARI0505I follows ARI0503E and provides a 3-digit number called an *SQLCODE* in its message text. For example, assume you receive the following messages:

```
ARI0503E An SQL error has occurred.
         SQL command begins properly but is incomplete.
ARI0505I SQLCODE = -106  SQLSTATE = 37501  ROWCOUNT = 0
ARI0504I SQLERRP: ARIXPA1 SQLERRD1: -150 SQLERRD2: 0
```

The SQLCODE provided in message ARI0505I is -106. Text for this SQLCODE begins on the second line of message ARI0503E and describes the cause of the error. If you want further explanation of the error, use the SQLCODE (in this example, -106) to view the online help information or to look up the explanation in the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals.

The SQLSTATE information provides a code for error conditions that are common across relational database products. For more information about SQLSTATE, refer to the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals.

The ROWCOUNT information is useful only for certain commands and is explained in the command descriptions. You can ignore the information in message ARI0504I, unless you are prompted to record it.

Entering Commands

DB2 Server for VSE

Figure 8 on page 12 shows a diagram of how ISQL divides your display.

```
                    OUTPUT AREA










-------------------------------------------------------------------------------
                    INPUT AREA
-------------------------------------------------------------------------------
                    STATUS AREA
```

Figure 8. DB2 Server for VSE & VM Display Format

The bottom line is the *status area*. Here, ISQL provides brief messages which indicate its current status during your terminal session. For example, whenever `Enter a command` appears, you know ISQL is ready to receive another command.

Note: For the remainder of this chapter, the general term *command* indicates ISQL commands, SQL statements and data.

You type commands in the input area, which is just above the status area. The input area consists of a single line and begins at the second-character position of the line. You do not have to move the cursor to this location; it is placed there by the database manager.

Although the --> cursor-movement key can be used to leave a blank space in the input line, it does not provide a blank character as does the space bar. Use the space bar to insert a blank character; use the cursor key to move the cursor.

The input area is also used by ISQL to provide the following message:

```
ARI7044I Command in progress. Terminal is now free.
```

This message, displayed when you have typed a command that is taking longer to execute than a preset amount of time, is meant for users involved with more than one CICS transaction. This message is only displayed if the ISQL user is connected to a local application server. ISQL is one of several CICS transactions available at your terminal. If you *are not* involved with multiple CICS transactions, ignore the message and wait for the command to complete. If you *are* involved with multiple CICS transactions and want to issue another CICS transaction while waiting for the command to be completed, this message indicates the terminal is free to do so. To enter another CICS transaction in response to this message, press CLEAR and type the desired CICS transaction identifier code. The transaction must not be pseudoconversational. For more information, see the *DB2*

Server for VSE System Administration and *DB2 Server for VM System Administration* manuals.

The output area displays information typed in the input area. It is also used to display any database manager responses to your input. Specific uses of the output area are discussed where appropriate in the manual.

DB2 Server for VM

A diagram of the way ISQL divides your display is shown in Figure 9.

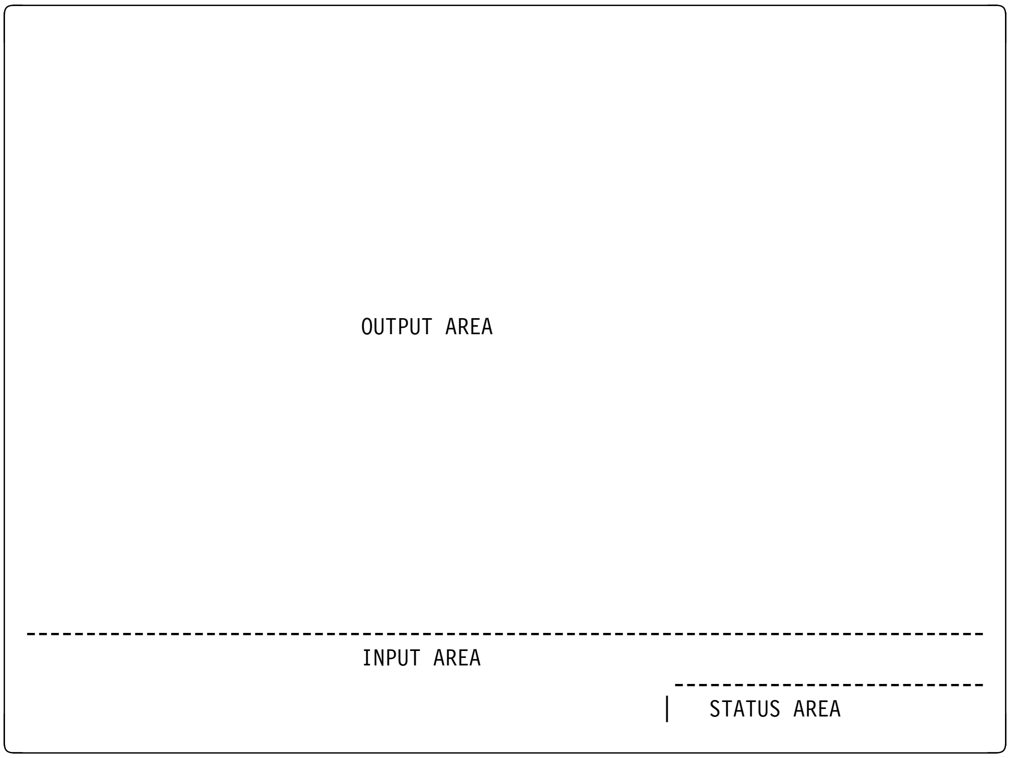


Figure 9. DB2 Server for VSE & VM Display Format

The last 21 characters of the bottom line are the status area. Here, the current VM system status is displayed during your terminal session. For example, when the `VM READ` message appears, you know that the system is ready to receive another command from you.

You type commands and data in the input area, which includes all of the line above the status area and the part of the next line that is to the left of the status area. The input area begins at the second character position of the display. You do not have to move the cursor to this location; it is placed there by the ISQL system.

The key marked `-->` only moves the cursor. This key does not insert blank characters into an input line. To insert a blank character, you must use the space bar.

The output area displays your commands and data after they are typed in the input area. It also displays ISQL status messages and any database manager responses to your input. Specific uses of the output area are discussed where appropriate in the manual.

ISQL status messages are displayed as informational or ARI-type messages. They appear on the last line of the output area.

The command line is usable at all times. You can type commands or statements even while others are being processed. For example, you can type a SELECT statement and then several FORMAT commands before the query results are displayed.

VM also permits you to stack commands. Commands are stored in the console stack; VM will then execute them one at a time.

Entering Commands While Viewing the Results of a Query

You can type any SQL statement or ISQL command while viewing the results of a query as long as that result does not originate from a query processed in an ISQL routine. Exceptions to this restriction are display commands, which you can use while viewing the results of a query being processed in a routine. (Routines are discussed in Chapter 9, "Creating and Using Routines" on page 125.)

If you do type a command, other than a display command, while viewing the results of a query being processed in a routine, you receive the following message:

```
ARI7956E Command failed.  This command is not valid
         while you view a query result from a routine.
```

The query result, because it is the result of a routine, is not affected.

If you are viewing a query result that is not from a routine, and you type an ISQL command or SQL statement that changes the current display, you receive the following message:

```
ARI7955I The system ended your query result to process your command.
```

In this situation, ISQL ends the current query display and processes the new command or statement.

If you are viewing a query result that is not from a routine, and you type a display command, the display command is processed.

Understanding ISQL Modes

The two modes in ISQL are *wait* and *display*. They provide different displays and, for particular commands and statements, they react differently. Wait mode is indicated by the VM READ displayed in the status area (lower right corner) of the display. In wait mode, you can enter any SQL statement and any ISQL command other than display commands. Each command or statement that you type is displayed line by line in the output area. From wait mode, you can query your application server. A query (a SELECT statement) puts the display into display mode.

In display mode, no message is displayed in the status area, and the entire output area displays the results of a query. From display mode, you can type any SQL statement or any ISQL command including ISQL display commands to move through the displayed data. You end display mode when you type END, an SQL statement other than SELECT, an ISQL command other than a display command, or an incorrect SELECT statement. In all instances wait mode is returned.

Using the Continuation Character

DB2 Server for VSE

Sometimes your input (commands, statements, or data) is too long to fit on the single input area line. When this happens, you can continue typing by using the continuation character, which is usually a hyphen. When the database manager is installed, the hyphen is the continuation character. The continuation character can be changed. For more information on the continuation character, see the SET command description in Chapter 13, “ISQL Commands” on page 203. This continuation character causes what you type to be redisplayed in the output area and frees the input area for more typing.

If a line ends in a complete word, leave a space after it, type the hyphen, and press ENTER. If you have to break a word at the end of a line, just type the hyphen without a space before it, and press ENTER.

The continuation character lets ISQL know that you have not finished with the command, and it responds with `continue command` in the status area. When you are finished typing, press ENTER. The entire command cannot exceed 2048 characters and the last line of the command must not end with a continuation character.

If the output area becomes full, you are prompted to clear it. Press CLEAR to clear both the input and output areas to allow the command in progress to continue. Incomplete portions of the command that were in the output area are stored in the SQL command buffer, but are not displayed. You do not have to repeat them. Everything in the input area is removed from the display when you press CLEAR.

DB2 Server for VM

Sometimes your input (commands, statements, or data) is too long to fit in the input area. When this happens, you can continue typing by using the continuation character, which is usually a hyphen. When the database manager is installed, the hyphen is the continuation character. The continuation character can be changed. For more information on the continuation character, see the SET command description in Chapter 13, “ISQL Commands” on page 203.

If a line ends with a complete word, leave a space after it, type the hyphen, and press ENTER. If you have to break a word at the end of a line, type the hyphen without a space before it, and press ENTER.

The continuation character indicates that you are not finished with the command. The system displays `VM READ` in the status area. It also displays the following message in the output area:

```
ARI7068I Your input is being continued. Type more input or press Enter.
```

Type the additional input for the command. When you are finished, press ENTER. The last line of the command must not end with a continuation character, and the entire command cannot exceed 2048 characters.

If the output area becomes full, you are prompted to clear it. Press CLEAR to clear both the input and output areas to allow the command in progress to continue. Incomplete portions of the command in the output area are stored in the SQL command buffer but are not displayed. You do not have to repeat them. Everything in the input area is removed from the display when you press CLEAR.

You can also press PA2 (Field Mark) to clear the output area. This clears only the output area and leaves anything that you have typed in the input area intact.

Correcting Typing Errors

DB2 Server for VSE

You can correct a typing error in the input area by backspacing and typing the correct characters before you press ENTER.

If you press ENTER before you notice the typing error, the command containing the typing error is displayed in the output area. It is no longer in the input area. You cannot backspace and retype information already in the output area.

To correct a mistake in a multiple-line command, type the following ISQL command at the beginning of the input area and press ENTER.

```
ignore
```

ISQL responds by telling you that it has ignored all lines previously entered in a multi-line input. The status area contains `Enter a new command`. You can then retype the command correctly.

You can also use the RETRIEVE function to review and correct any typed information. ISQL saves the typed lines.

To use the RETRIEVE function, press PF12 (or PF24).

This function retrieves the last line that you typed and redisplays it in the input area for review and correction. The cursor is positioned at the end of the displayed line. You can continue to press PF12 (or PF24) until the command you want to change is redisplayed in the input area. Then, you can make any necessary corrections and press ENTER.

Each time you press F12, ISQL retrieves another line and redisplays it in the input area. After the earlier line is retrieved, command retrieval begins again with the most recent command typed. ISQL retains a varying number of your commands depending on their size. The shorter the commands, the more ISQL retains.

DB2 Server for VM

You can correct a typing error in the input area by backspacing and typing the correct characters before you press ENTER.

If you press ENTER before you notice the typing error, the command or statement containing the typing error is displayed in the output area. It is no longer in the input area. You cannot backspace and retype information already in the output area.

To correct a mistake in a multiple-line command, type the following ISQL command at the beginning of the input area and press ENTER.

```
ignore
```

ISQL responds by telling you that it has ignored all lines previously entered in a multi-line input. The status area contains `VM READ`. You can then retype the information correctly.

You can also use the CMS RETRIEVE function to review and correct typed information. CMS saves the lines that you have typed.

To use the RETRIEVE function, press PF12.

This function retrieves the last line that you typed and redisplays it in the input area for your review and, if necessary, correction. The cursor is positioned at the end of the displayed line. You can continue to press PF12 (or PF24) until the command that you want to change is redisplayed in the input area. Then you can make any necessary corrections and press ENTER to start the command.

Each time that you press PF12, CMS retrieves another line and redisplays it in the input area. After the earlier line is retrieved, command retrieval begins again with the most recent command typed. CMS retains a varying number of your commands, depending on their size. The shorter the commands, the more CMS retains.

The retrieve key function operates as if ISQL has two command stacks, one for query commands (commands related to a SELECT statement) and one for nonquery commands. Query commands typed *during* the query cannot be retrieved after the query is finished. For more information on the RETRIEVE function, see “Retrieving and Correcting SQL Lines” on page 87.

Using ISQL on a Non-DB2 Server for VM Application Server (VM Only)

With the implementation of the Distributed Relational Database Architecture™ (DRDA) protocol, you can use ISQL to access data on non-DB2 Server for VM application servers that support the DRDA protocol and on which ISQL is loaded. If you plan to use ISQL to access data on a non-DB2 Server for VM application server, you should check with your System Administrator to see whether your application requester and the non-DB2 Server for VM application server are set up for access using the DRDA protocol.

Using ISQL on a Remote Application Server (VSE Only)

With the implementation of the Distributed Relational Database Architecture (DRDA) protocol, you can use ISQL to access data on remote DRDA application servers that support the DRDA protocol and on which ISQL is loaded. If you plan to use ISQL to access data on a remote DRDA application server, you should check with your System Administrator to see whether your application requester and the remote DRDA application server are set up for access using the DRDA protocol.

Stopping ISQL

DB2 Server for VSE

To stop communication with the application server through ISQL, type the following command in the input area and press ENTER.

```
exit
```

This completes your first ISQL session.

DB2 Server for VM

To stop communication with the application server through ISQL, type the following command in the input area:

```
exit
```

Press ENTER.

Entering Commands While Viewing the Results of a Query

You can type any ISQL command or SQL statement while viewing the results of a query only if the result is not from a query being processed in an ISQL routine. Exceptions to this restriction are display commands, which you can use while viewing the results of a query being processed in a routine. (For information about routines refer to Chapter 9, "Creating and Using Routines" on page 125.)

If you do type a command, other than a display command, while viewing the results of a query being processed in a routine, you receive the following message:

```
ARI7963E Command failed. This command is not valid while you view
         a query result from a routine or an EXEC.
```

Because the query result is the result of a routine, it is not affected.

If you are viewing a query result that is not from a routine, and you type an ISQL command or SQL statement that changes the current display, you receive the following message:

```
ARI7955I The system ended your query result to process your command.
```

In this case, ISQL ends the current query display and processes the new command or statement.

If you are viewing a query result that is not from a routine, and you type a display command, the display command is processed.

CHARNAME and DBCS Options

ISQL queries the following system catalog tables at initialization:

1. SYSTEM.SYSOPTIONS catalog - to retrieve information such as the default setting for DBCS and CHARNAME. In DB2 Server for VSE Version 6 Release 1, this information is retrieved from the SQLGLOB file. The user DBCS and CHARNAME SQLGLOB values become the default setting for DBCS and CHARNAME, if they exist. Otherwise, the global DBCS and CHARNAME SQLGLOB values become the default setting for DBCS and CHARNAME.

   ISQL uses the user CHARNAME to get the folding table to fold input from the terminal from lowercase to uppercase. However, for CCSID data conversion, ISQL uses the global CHARNAME.

2. SYSTEM.SYSCHARSETS catalog - to retrieve the CHARTRANS information corresponding to the default CHARNAME. In DB2 Server for VSE Version 6 Release 1, this information is retrieved from the ARISSCRD phase file, if it exists. If not, the system defaults are supplied.

Chapter 2. Querying Tables

Querying table data is the most common activity performed by the database manager.

The topics that follow describe query techniques that use data from the sample tables in Appendix B, "Sample Tables" on page 269. Before continuing, check with the persons responsible for the system in your organization to ensure the sample tables are stored in the system for your use. If they are, you can use them to practice commands and statements as you learn them. Having access to the sample tables is not absolutely necessary, but being able to work with them from your display terminal is of considerable help as you learn to use the system.

Selecting Particular Columns

Use the interactive SELECT statement to query data in a table. With this statement, you *select columns from a table*.

The SELECT statement requires two parts:

SELECT clause Lists the columns you want to select.

FROM clause Names the table from which to select the columns.

These two parts, or clauses, of the SELECT statement are always needed. The SELECT clause is always first, immediately followed by the FROM clause.

Suppose that you want to select all the columns from the DEPARTMENT table. Start ISQL, and type a SELECT statement in the input area as shown in Figure 11 or Figure 10. List the columns you want and name the table that contains them.

Note: ISQL recognizes (by default) uppercase and lowercase letters as identical in commands. Also by default, all character data in a table is stored as uppercase. Your query on the row containing the word *James*, for example, selects the row containing *JAMES* in uppercase, even if you typed *james* in lowercase in your query. These case defaults can be changed using the SET CASE command, which is discussed in detail in Chapter 13, "ISQL Commands" on page 203.

```
select deptno,deptname,mgrno,admrdept from department
                                                          Enter a command
```

Figure 10. DB2 Server for VSE SELECT Statement Entered from Wait Mode

```
select deptno,deptname,mgrno,admrdept from department
                                                            VM READ
```

Figure 11. DB2 Server for VM SELECT Statement Entered from Wait Mode

You have just typed your first query.

Note: Naming conventions for columns, tables, and other SQL features are in the *DB2 Server for VSE & VM SQL Reference* manual.

Now press ENTER.

The application server retrieves the data, and ISQL displays your query result in the output area. The display then looks like Figure 12.

```
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT
------  -----------------------------------  ------  --------
A00     SPIFFY COMPUTER SERVICE DIV.         000010  A00
B01     PLANNING                             000020  A00
C01     INFORMATION CENTER                   000030  A00
D01     DEVELOPMENT CENTER                   ?       A00
D11     MANUFACTURING SYSTEMS                000060  D01
D21     ADMINISTRATION SYSTEMS               000070  D01
E01     SUPPORT SERVICES                     000050  A00
E11     OPERATIONS                           000090  E01
E21     SOFTWARE SUPPORT                     000100  E01
* End of Result ************** 9 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 12. Display Mode Showing Query Result

The SELECT clause causes the DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT columns to be selected, and the FROM clause names the DEPARTMENT table as the table from which to select the columns. Here you selected all four columns in the table, but you can select any number of columns (up to the maximum of 45). When selecting more than one column, use commas to separate the column names.

The sequence in which the columns are displayed is the same sequence in which the column names appear in the SELECT clause. You control the left-to-right order of the columns of your query results by specifying the sequence of column names.

The displayed length attribute of VARCHAR columns such as DEPTNAME is determined by the default VARCHAR length set in your profile. The system default length for VARCHAR columns is 20 characters. Only the first 20 characters are displayed. You can change the default length by using the SET command. Because the default length for VARCHAR columns can be different, you may see more or fewer characters on your display of VARCHAR columns than are shown in this manual.

You may have seen the following message on your display just before the query result:

```
ARI7960I The query cost estimate for this SELECT statement is 1.
```

The *cost estimate* value for this SELECT statement is also displayed in the last row of output at the end of the query result.

This information is useful for estimating the time needed to obtain query results. Larger cost-estimate values suggest that processing takes longer. The information is provided *before* the query result is displayed so that you can cancel the request if you think it will take too much time. The information is also provided *after* the query result is displayed, because the query result may have been displayed before you read message ARI7960I, or the message may have been suppressed. For additional information on the SET COSTEST command refer to Chapter 13, “ISQL Commands” on page 203.

The cost estimate value is not a unit of time such as seconds. The value becomes more useful as you enter more SELECT statements and become acquainted with the processing times for the cost estimate values.

DB2 Server for VSE

Notice the status area. It contains `Enter a DISPLAY command`.

You can manipulate query results shown on your display by using ISQL display commands. The only display command that you need to know now is the END command, which the following sections discuss. Other display commands are described in subsequent sections of this manual.

Ending a Query Display

Always end a query result when you are finished with it (so that ISQL performance for other users is not affected). The END command removes your query result from the display. Type the following and press ENTER:

```
end
```

DB2 Server for VM

ISQL returns to wait mode.

Selecting All Columns

In the previous examples, information was selected by listing the names of the columns desired. When all the columns of a table are to be selected, you can substitute an asterisk for the column names. If the table has more than 45 columns, only the first 45 columns are displayed. To display more than 45 columns, see the SELECT statement discussion in the *DB2 Server for VSE & VM SQL Reference* manual.

Query statements can be typed on one line by omitting the continuation character, or they can be typed on several lines by incorporating the continuation character. For example, type the following query in the input area and press ENTER. This query comprises two separate statements that can be typed on the same line because the continuation character is omitted.

```
select * from department
```

Your display should look like Figure 13 on page 24. Type the END command to end the query result, and then type the following and press ENTER:

```
select * -
```

A continuation character was used. Now type the following and press ENTER:

```
from department
```

This query, using a continuation character, produces the same display as before, as shown in Figure 13 on page 24. Continuation characters are used in most of the examples and exercises, because they allow the entry of lengthy or complex queries.

```
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT
------  -----------------------------------  ------  --------
A00     SPIFFY COMPUTER SERVICE DIV.         000010  A00
B01     PLANNING                             000020  A00
C01     INFORMATION CENTER                   000030  A00
D01     DEVELOPMENT CENTER                   ?       A00
D11     MANUFACTURING SYSTEMS                000060  D01
D21     ADMINISTRATION SYSTEMS               000070  D01
E01     SUPPORT SERVICES                     000050  A00
E11     OPERATIONS                           000090  E01
E21     SOFTWARE SUPPORT                     000100  E01
* End of Result *** 9 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 13. A Query That Selects All Columns

When you use an asterisk, the order in which the columns are displayed is the order in which they appear in the table. Type the END command to end the query result.

EXERCISE 1 (Answers are in Appendix A, Answers to the Exercises, on page 261.)

Create and type a statement for each of the following:

1. Select the column called ACTNO from the ACTIVITY table.
2. Select the DEPTNO, RESPEMP, and PRSTAFF columns from the PROJECT table.
3. Select all the columns from the PROJ_ACT table.

Selecting Particular Rows

In the previous example, you selected all the rows in the table. Suppose you want the rows of the EMPLOYEE table with an entry of *James* in the FIRSTNME column. To extract this conditional information, you must add a WHERE clause to the SELECT statement. The form is:

```
►──WHERE──search_condition──────────────────────────────────────►
```

You can also specify a search condition for groups of rows that satisfy a particular condition by using a HAVING clause. A HAVING clause contains one or more group-qualifying predicates connected by AND and OR operators. (Predicates are discussed later in this chapter; operators are discussed in the next section.) The form is:

```
►──HAVING──search_condition──────────────────────────────────────────►
```

search_condition
is one or more conditions (or predicates) to apply in selecting, updating, or deleting data. These conditions are basically a comparison between two expressions.

Introducing Expressions and Operators

An expression refers to a column, a constant, an SQL special register, a column function, a scalar function, an arithmetic expression, a character expression, or a labeled duration. Expressions are connected by one or more of the following arithmetic operators:

+ (add)

– (subtract) (multiply)

/ (divide)

The following are examples of valid arithmetic expressions:

bonus + 200
salary * prstaff
7000 / 3.1416

Each condition can include one or more of the following Boolean (or logical) operators:

Operator	**Example**
AND	where salary > 30000 AND sex = 'm'
OR	where projno = 'AD3100' OR projno = 'AD3111'
NOT	where NOT(projno = 'AD3100 OR projno = 'AD3111')

Each condition can also include one of the following comparison operators:

=	(equal to)
<>	(not equal to)
>	(greater than)
>**=**	(greater than or equal to)
<	(less than)
<**=**	(less than or equal to)

You can use these operators to compare one item with another. For example, using the DEPARTMENT table in the sample tables, you can use the following search condition to select the row whose department number is A00:

```
select * -
from department -
where deptno = 'A00'
```

The specified value for DEPTNO is enclosed in single quotation marks because the column is defined as character-type data. No quotation marks are needed for numeric-type data.

Type the following statement in the input area and press ENTER.

```
select * -
from employee -
where firstnme = 'JAMES'
```

The query result is displayed as shown in Figure 14.

```
EMPNO   FIRSTNME      MIDINIT  LASTNAME        WORKDEPT  PHONENO  HIREDATE
------  ------------  -------  --------------- --------  -------  ----------
000190  JAMES         H        WALKER          D11       2986     1974-07-26
000230  JAMES         J        JEFFERSON       D21       2094     1966-11-21
* End of Result *** 2 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 14. Result of a Query Using Expressions and Operators

When used, a WHERE clause always follows the FROM clause. The WHERE clause in this example specifies that only the rows that contain *JAMES* in their FIRSTNME column are to be selected. Again, *JAMES* must be enclosed in single quotation marks to show that it refers to character data; do not enclose numeric data in quotation marks.

You can now end the query result. This time, however, press PF3 instead of typing the END command. PF3 enters an END command and you can use it whenever you want to end a query result.

Preventing the Selection of Duplicate Rows

Sometimes you want to prevent the selection of rows that duplicate others you have already selected. For example, if you want to know the department numbers in the PROJECT table, type the following statement:

```
select deptno -
from project
```

Press ENTER.

This displays the result in Figure 15 on page 27.

```
DEPTNO
------
B01
C01
C01
D01
D01
D11
D11
D11
D11
D21
D21
D21
D21
E01
E01
E11
E21
E21
E21
E21
* End of Result *** 20 Rows Displayed ***Cost Estimate is 1********************
```

Figure 15. A Query Result with Duplicate Rows

The result shows the department numbers, but particular department numbers are repeated. Press PF3 to end the query result.

To eliminate repeated numbers, modify the SELECT clause as follows:

```
select distinct deptno -
from project
```

Press ENTER.

This produces the result shown in Figure 16.

```
DEPTNO
------
B01
C01
D01
D11
D21
E01
E11
E21
* End of Result *** 8 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 16. A Query Result without Duplicate Rows

Press PF3 to end the query result.

Using the AND and OR Operators

It is often necessary to select rows of a table based on more than one search condition. Multiple conditions are connected by the logical operators AND or OR.

The difference between the AND and OR operators is that with AND *all* conditions connected by the AND operator must be met for the row to be selected, but with OR, if *any* of the conditions are met, the row is selected.

The following example illustrates the AND operator. This query searches the EMPLOYEE table and selects only female employees who earn a salary greater than $30000.

```
select * from employee -
where salary > 30000 and sex = 'f'
```

The following example illustrates the OR operator. The SQL UPDATE statement changes the bonus to $800 for employees whose education level is greater than 18 or whose job is *analyst*.

```
update employee -
set bonus = 800 -
where edlevel > 18 or job = 'analyst'
```

Using the NOT Operator

You can use the NOT operator with an expression to indicate a condition to be met when you are selecting rows. For example, the following statement selects the rows from the EMP_ACT table that have employee project time, but that do not have an activity number of 90:

```
select * -
from emp_act -
where emptime > 0 and not actno = 90
```

Note: You can also type the WHERE clause as:

```
where emptime > 0 and actno <> 90
```

(*not actno = 90* is the same as *actno* <> *90*).

Grouping Search Conditions

You can use parentheses to group conditions to give higher precedence to the logic within the parentheses. For example, you can use the following search conditions to create a *view* (discussed in Chapter 10, “Using Additional Query Techniques” on page 135) of the rows of the DEPARTMENT table where the administrative reporting department is A00 or D01, and the department number is A00:

```
create view rpt -
as select * from department -
where deptno = 'a00' -
and (admrdept = 'a00' or admrdept = 'd01')
```

If this search condition is used without the parentheses, a view is created for the first, fifth, and sixth rows of the DEPARTMENT table, instead of just for the first. This is because, without the parentheses, the AND operator would take precedence over the OR operator.

Additional Examples

To select all the rows from the PROJ_ACT table for project number AD3100, and all the rows for project number AD3111 except the row for activity number 60, type the following statement:

```
select * -
from proj_act -
where projno = 'AD3100' or projno = 'AD3111' -
and not actno = 60
```

Note: You can also type the WHERE clause as:

```
where projno='AD3100' or projno='AD3111' and actno<>60
```

Press ENTER.

This clause produces the result shown in Figure 17.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3100      10     0.50  1982-01-01  1982-07-01
AD3111      70     1.50  1982-02-15  1982-10-15
AD3111      70     0.50  1982-03-15  1982-10-15
AD3111      80     1.25  1982-04-15  1983-01-15
AD3111      80     1.00  1982-09-15  1983-01-01
AD3111     180     1.00  1982-10-15  1983-01-15
* End of Result *** 6 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 17. A Query Result Using NOT and Grouping Search Conditions

The WHERE clause in Figure 17 can also be written using parentheses:

```
where projno = 'AD3100' -
or (projno = 'AD3111' and not actno = 60)
```

Parentheses group multiple search conditions to form one search condition; the WHERE clause shown above contains two search conditions:

- projno='AD3100'
- projno='AD3111' and not actno=60

Press PF3 to end the query result.

The search conditions in a WHERE clause can be modified merely by a change in the location of the parentheses. For example, to omit the rows for activity number 60 from both projects in the above example, type the following statement:

```
select * -
from proj_act -
where (projno = 'AD3100' or projno = 'AD3111') -
and not actno = 60
```

Press ENTER.

The query displays the result shown in Figure 18 on page 30.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3100      10     0.50  1982-01-01  1982-07-01
AD3111      70     1.50  1982-02-15  1982-10-15
AD3111      70     0.50  1982-03-15  1982-10-15
AD3111      80     1.25  1982-04-15  1983-01-15
AD3111      80     1.00  1982-09-15  1983-01-01
AD3111     180     1.00  1982-10-15  1983-01-15
* End of Result *** 6 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 18. A Query Result When Search Condition is Modified by Parentheses

Press PF3 to end the query result. (By now you are probably getting used to using the END command (PF3 key); it is mentioned only occasionally in the remainder of the manual.)

EXERCISE 2 (Answers are in Appendix A, Answers to the Exercises, on page 261.)

Create and type a statement for each of the following:

1. Select the information in the LASTNAME, WORKDEPT, and PHONENO columns of the EMPLOYEE table for the rows where the salary is greater than $50000.
2. Select the department number, department name, manager number, and senior department number information from the DEPARTMENT table for any department that reports to department A00.
3. Select the activity numbers from the PROJ_ACT table that need 0.75 or more mean number of employees to staff the activity.
4. Select all the information in the PROJ_ACT table that applies to project numbers AD3100, IF1000, or MA2111.

Selecting Rows by Using Special Registers

You can use special registers whenever you use an expression in statements such as SELECT or UPDATE. The special registers are:

CURRENT DATE Returns the current date in the local time zone.

CURRENT SERVER Returns the name of the application server to which your terminal is connected.

CURRENT TIME Returns the current time in the local time zone.

CURRENT TIMESTAMP Returns the current timestamp in the local time zone.

CURRENT TIMEZONE Returns a signed time duration as a DEC(6,0) number containing the local time-zone value.

USER Returns your authorization ID.

The value stored in the table for CURRENT DATE is a date value (years, months, and days). The value stored in the table for CURRENT TIME is a time value (hours, minutes, and seconds). The CURRENT TIMESTAMP stores a combination of date and time values (combining date and time elements, which include microseconds).

You can use CURRENT SERVER to display the name of the application server to which your terminal is connected. In the following example, CURRENT SERVER displays the name of the application server on which the table SAMP1 resides.

```
select current server,id,indate -
from samp1
```

You can use special registers such as CURRENT TIME and CURRENT TIMESTAMP to timestamp your tables. The sample tables do not contain time values, but suppose you query a table called OPERATIONS that has a TIME or TIMESTAMP column as follows:

```
select * from operations -
where end_time = '16:30'
```

This SELECT statement results in a query display of all rows from OPERATIONS where the value in the END_TIME column is *16:30* (16 hours and 30 minutes). If any of the END_TIME entries are generated by a special register, those entries include the seconds part of the time value as well, and may include the microseconds if the column is defined as a TIMESTAMP column.

When you query rows whose time values have only hours and minutes, the rows with seconds values are bypassed. Your query result is incomplete, and essential rows may be missing. To include the missing rows, word the SELECT statement more carefully as in the following example:

```
select * from operations -
where hour(end_time) = 16 -
and minute(end_time) = 30
```

This statement produces a query result that includes all rows with time values of 16 hours, 30 minutes, and anywhere from 0 to 59 seconds.

You can also have problems of consistency and accuracy when dealing with datetime values from *different* time zones, or daylight-saving-time values that are incompatible with non-daylight-saving-time values from the *same* time zone.

The CURRENT TIMEZONE allows easy conversion of a local time into Greenwich Mean Time (GMT) by subtracting CURRENT TIMEZONE from local time. The CURRENT TIMEZONE can be subtracted from a TIME or a TIMESTAMP data type. This value is a negative time value if you are west of the GMT zone, and a positive value if you are east of the GMT zone.

The value of each of the four special registers that deal with time is based on the Time-of-Day (TOD) clock reading performed during the processing of the statement containing the particular special register.

You can use special register USER in the definition of a search condition or in a select list. The database manager replaces USER with your authorization ID when it processes the search condition. For example, if your authorization ID is SMITH and the following search condition is specified:

```
where name = USER
```

The database manager searches for name = 'SMITH'.

Selecting Rows That Contain a Matched Value

With the IN predicate you can select rows that contain a particular value (or character combination) in a certain column by matching the value with a value or a list of values (or character combinations) that you specify.

With the IN predicate you can quickly compare the value of an expression with an item or a list of items that you specify. The format of an IN predicate is:

```
►──expression──┬─────┬──IN──┬─(subselect)──────────────────────┬──────────►
               └─NOT─┘      │     ┌─,──────────────────┐       │
                            └─(───▼─┬─constant─────────┬─┴──)──┘
                                    └─special_register─┘
```

The condition is satisfied if the expression is equal to one or more items or, if the NOT option is used, is not equal to any item. The following example illustrates this type of condition:

```
deptno in ('D01','B01','C01')
```

The list can be formed by using a subquery. Subqueries are discussed in Chapter 10, “Using Additional Query Techniques” on page 135.

To select rows from the DEPARTMENT table for manager numbers 10, 20, and 30, you could type the following statement:

```
select * -
from department -
where mgrno = '000010' -
or mgrno = '000020' -
or mgrno = '000030'
```

You can also use the IN predicate as part of the WHERE clause to obtain the same result by typing:

```
select * -
from department -
where mgrno in ('000010','000020','000030')
```

Either query would present the display in Figure 19.

```
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT
------  -----------------------------------  ------  --------
A00     SPIFFY COMPUTER SERVICE DIV.         000010  A00
B01     PLANNING                             000020  A00
C01     INFORMATION CENTER                   000030  A00
* End of Result *** 3 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 19. A Query Result from Using the IN Predicate

Selecting Rows That Contain Values within a Range

Sometimes you must select data that falls within a range; for example, you want to select the rows from the EMPLOYEE table where the salary is between $30000 and $40000. Obtain this information by typing:

```
select workdept,phoneno,hiredate,job,edlevel,sex,birthdate,salary -
from employee -
where salary between 30000 and 40000
```

This produces the display in Figure 20.

```
WORKDEPT  PHONENO  HIREDATE    JOB       EDLEVEL  SEX  BIRTHDATE        SALARY
--------  -------  ----------  --------  -------  ---  ----------  -----------
C01       4738     1975-04-05  MANAGER        20  F    1941-05-11     38250.00
D11       6423     1973-09-14  MANAGER        16  M    1945-07-07     32250.00
D21       7831     1980-09-30  MANAGER        16  F    1953-05-26     36170.00
* End of Result *** 3 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 20. A Query Result from Using the BETWEEN Predicate

Using the BETWEEN predicate, you can determine whether a value falls within a specified range of values. The format of a BETWEEN predicate is:

```
►──expression_1──┬─────┬──BETWEEN──expression_2──AND──expression_3──────►
                 └─NOT─┘
```

The condition is satisfied if *expression_1* is equal to or greater than *expression_2*, and equal to or less than *expression_3*. These three expressions are standard expressions containing column names and constants.

The BETWEEN predicate can be modified by NOT. With the NOT option, the condition is satisfied if *expression_1* is less than *expression_2* or greater than *expression_3*.

Selecting Rows That Contain a Particular Character Combination

Rows can be selected that contain a particular combination of characters in columns containing character data.

Evaluating Character String Expressions

The database manager observes the following rules when evaluating character-string expressions:

1. When comparing two character strings, alphanumeric ordering is used. The number 9 is greater than 8 and so on to 0. The number 0 is greater than the uppercase Z, which is greater than Y and so on to A. The character A is greater than the lowercase z, which is greater than y and so on to a. The character a is greater than the special characters.
2. When comparing two character strings of fixed length, or when comparing a fixed-length string and a varying-length string, the database manager pads the shorter string on the right with blanks until it equals the length of the longer

string; then it performs the comparison. For example, if *SMITH* is compared with a 10-character string, the database manager pads it with blanks like this:

```
'SMITH     '
```

3. Two VARCHAR strings must match in content, not length, to be considered equal. For example, `'SMITH '` is the same as `'SMITH'`.

   Trailing blanks, however, are significant when you compare LONG VARCHAR strings or LONG GRAPHIC strings. Trailing blanks are also significant when you use the LIKE predicate.

4. A VARCHAR field that has a length of zero can be represented in a search condition by two single quotation marks without a blank between them.

For more information and examples, see “Selecting Rows That Contain Null Values” on page 37.

Using the LIKE Predicate

The LIKE predicate searches for character string data that *partially* matches a particular string. Its format is:

```
►──column_name──┬─────┬──LIKE──┬─string_constant─┬──┬────────────────────────────────┬──────────►
                └─NOT─┘        └─USER────────────┘  └─ESCAPE──escape_character─┘
```

As indicated in the syntax diagram, NOT can modify LIKE.

Note: You *cannot* use LIKE with a date or time value.

The specified column must be of fixed-length or varying-length character or graphic data type. Datetime columns and numeric type columns are not permitted. The *string_constant* can contain any character string, with special meanings reserved for the underscore (_) and percent (%) characters.

Defining the Underscore and Percent Characters

The underscore character represents any single character. The percent character represents any string of zero or more characters. These two special characters can be used in any combination in the *string_constant*. The following examples illustrate the use of the LIKE predicate, the underscore character, and the percent character:

```
name like '%anne%'
```

This example searches for any name that contains the word *anne*, for example, *LIZANNE*, *ANNETTE*, or *ANNE*.

The following example searches for any 3-character description that has an *N* as the second character.

```
description like '_n_'
```

To satisfy this pattern, a data value must be 3 characters long. Its data type can be either CHAR or VARCHAR.

The following example uses both special characters in the same *string_constant*.

```
name like '_e_e%'
```

This example searches for any name that is a minimum of 4 characters and has E as the second and fourth characters.

Note: The character string can be longer than 4 characters.

Using the Percent Character

To select from the ACTIVITY table the activity keyword and activity description of those activities whose descriptions are maintenance-related, type the following SELECT statement:

```
select actkwd,actdesc -
from activity -
where actkwd like '%maint%'
```

This produces the result in Figure 21.

```
ACTKWD  ACTDESC
------  --------------------
MAINT   Maint software sys
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 21. A Query Result from Using the Percent Character

Using the Underscore Character

Select the rows from the PROJ_ACT table that contain a project number of 6 characters and starts with the an A:

```
select * -
from proj_act -
where projno like 'A _ _ _ _ _'
```

Note: Do *not* leave blanks either between the underscores, or between the first underscore and the letter A.

The query displays the result in Figure 22 on page 36.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3100      10     0.50  1982-01-01  1982-07-01
AD3110      10     1.00  1982-01-01  1983-01-01
AD3111      60     0.80  1982-01-01  1982-04-15
AD3111      60     0.50  1982-03-15  1982-04-15
AD3111      70     1.50  1982-02-15  1982-10-15
AD3111      70     0.50  1982-03-15  1982-10-15
AD3111      80     1.25  1982-04-15  1983-01-15
AD3111      80     1.00  1982-09-15  1983-01-01
AD3111     180     1.00  1982-10-15  1983-01-15
AD3112      60     0.75  1982-01-01  1982-03-15
AD3112      60     0.50  1982-02-01  1982-03-15
AD3112      60     0.75  1982-12-01  1983-01-01
AD3112      60     1.00  1983-01-01  1983-02-01
AD3112      70     0.75  1982-01-01  1982-10-15
AD3112      70     0.50  1982-02-01  1982-03-15
AD3112      70     1.00  1982-03-15  1982-08-15
AD3112      70     0.25  1982-08-15  1982-10-15
AD3112      80     0.35  1982-08-15  1982-12-01
AD3112      80     0.50  1982-10-15  1982-12-01
AD3112     180     0.50  1982-08-15  1983-01-01
AD3113      60     0.75  1982-03-01  1982-10-15
AD3113      60     1.00  1982-04-01  1982-09-01
```

Figure 22. A Query Result from Using the Underscore Character

Using an Escape Character

If the underscore or percent characters are in the string constant, you can define an escape character that instructs the database manager not to treat them as special characters but as part of the string.

The following example illustrates the definition of an escape character:

```
name like '+%+_vacancy %' ESCAPE '+'
```

The ESCAPE `'+'` defines + as the escape character and indicates a search for a string that begins with the characters %_VACANCY.

The +% and the +_ at the beginning of the string indicate that the percent and underscore characters are part of the string constant. Because no + appears before the second % in the example, % accepts *any* character string to the right of %_VACANCY; for example, %_VACANCY RATE.

If you use the special register USER in a character-string search, it is interpreted as a CHAR(8) string whose value is the authorization ID of the user currently connected. The following example illustrates what you would type:

```
name like user
```

The special register USER is further discussed in “Selecting Rows by Using Special Registers” on page 30.

Note: To prepare for the following exercise, type SET CASE STRING on the ISQL command line, and press ENTER. The SET CASE STRING setting and other SET commands are explained in Chapter 13, "ISQL Commands" on page 246.

EXERCISE 3 (Answers are in Appendix A, Answers to the Exercises, on page 261.)

Create and type a statement for each of the following:

1. Select the activity numbers from the PROJ_ACT table that require a mean number of employees between 0.75 and 1.25.
2. Select the project number and activity start date from the PROJ_ACT table for project numbers AD3100, IF1000, and MA2111.
3. Select the activity number from the ACTIVITY table for those activities that have the word DATA in their description.

Selecting Rows That Contain Null Values

A null represents an undefined value. A null differs from a blank in that a blank in a table indicates that the value is blank for that position. A null indicates that no value is defined for this position. A null value is represented in an ISQL query display by a control character (usually a question mark).

The IS NULL predicate either searches for null values in a table, or excludes null values from the result of a query. The format of an IS NULL predicate is:

```
►──column_name──IS──┬───────┬──NULL──────────────────────────────────►
                    └─NOT───┘
```

A row of a table satisfies this condition if the value in the designated column is (or is not) null.

The following statement retrieves those rows from the DEPARTMENT table that have a null value in their MGRNO field:

```
select * -
from department -
where mgrno is null
```

If you want to search for a null value among other items, you must put the null value in a separate search condition rather than as a list item. For example, the following statement retrieves those rows from the PROJECT table whose PRSTAFF field contains 1, 2, or a null value:

```
select * -
from project -
where prstaff in (1,2) -
  or prstaff is null
```

when you retrieve data from tables that contain null values, you may not retrieve all the data you want. Suppose you query a table as follows:

```
select * -
from proj_act -
where acstaff <> .50
```

The ACSTAFF column was defined to allow nulls. The query result only provides those rows where the ACSTAFF column is not assigned the value of .50. You *would not* see rows that contain a null value in the ACSTAFF column. To view these rows, as well as those with a value other than .50, type:

```
select * -
from proj_act -
where acstaff <> .50 or acstaff is null
```

Selecting Rows That Satisfy a Calculated Condition

A search condition can also contain arithmetic operators that allow you to select rows based on calculated values. For example, if you want to identify the employees in the EMPLOYEE table that earn more than $40000 when their salary and bonus are added, type:

```
select empno,salary -
from employee -
where salary + bonus > 40000.00
```

This presents the result in Figure 23.

```
EMPNO        SALARY
------  -----------
000010     52750.00
000110     46500.00
000020     41250.00
000050     40175.00
* End of Result *** 4 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 23. A Query Result That Satisfies a Calculated Condition

The arithmetic operations that can be performed are:

+ (add)
– (subtract)
* (multiply)
/ (divide)

More than one arithmetic operator can appear in a search condition, and parentheses can be used in the same way they are used in any arithmetic expression.

Note: The conversion processes needed to use operands of different data types in calculations are described in the *DB2 Server for VSE & VM SQL Reference* manual.

Selecting Rows Using Durations

A duration is an interval of time expressed in the form of a labeled duration, or a date, time, or timestamp duration.

Labeled Durations

A labeled duration represents any number of years, months, days, hours, minutes, seconds, or microseconds. Such an expression lets you isolate parts of a date or time value and accomplish tasks, such as adding 4 days to a date value in a table. The number can be either a numeric constant, a column-name, a function, or a numeric expression in parentheses. The number is then converted as if it were assigned to a DEC(15,0) number. Fractional durations are truncated to the whole number. The unit is expressed by a keyword following the number.

The keywords are:

YEAR(S)
MONTH(S)
DAY(S)
HOUR(S)
MINUTE(S)
SECOND(S)
MICROSECOND(S).

An example of a labeled duration is:

```
start_date + 120 days
```

The labeled duration, *120 days*, results in 120 days being added to the starting date.

Date Durations

A date duration represents a number of years, months, and days. It is expressed as a DEC(8,0) number. The interpretation of the 8 digits is *yyyyxxdd*, where *yyyy* is the number of years, *xx* is the number of months, and *dd* is the number of days.

An example of a date duration is:

```
date1 - date2
```

where both *date1* and *date2* are defined as date values.

Time Durations

A time duration represents a number of hours, minutes, and seconds. It is expressed as a DEC(6,0) number. The interpretation of the 6 digits is *hhmmss*, where *hh* is the number of hours, *mm* is the number of minutes, and *ss* is the number of seconds.

An example of a time duration is:

```
time1 + time2
```

where both *time1* and *time2* are defined as time values.

Timestamp Durations

A timestamp duration represents an amount of time. It is expressed as a DEC(20,6) number. The interpretation of the 20 digits is *yyyyxxddhhmmsszzzzzz*, where *yyyy* is the number of years, *xx* is the number of months, dd is the number of days, *hh* is the number of hours, *mm* is the number of minutes, *ss* is the number of seconds, and *zzzzzz* is the number of microseconds.

An example of a timestamp duration is:

```
timestamp1 - timestamp2
```

where both *timestamp1* and *timestamp2* are defined as timestamp values.

Using a Labeled Duration

Suppose you want to select the rows from the EMPLOYEE table where the time between birth date and hiring date is less than 25 years. Type the following SELECT statement:

```
select hiredate,job,edlevel,sex,birthdate,salary,bonus -
from employee -
where birthdate + 25 years > hiredate
```

The result is displayed in Figure 24.

```
HIREDATE    JOB       EDLEVEL  SEX  BIRTHDATE        SALARY        BONUS
----------  --------  -------  ---  ----------  -----------  -----------
1963-12-05  CLERK          14  M    1942-10-18     29250.00       600.00
1972-02-12  DESIGNER       16  M    1947-05-17     25280.00       500.00
1977-10-11  DESIGNER       17  F    1955-04-12     22250.00       400.00
1973-07-07  DESIGNER       17  F    1949-02-21     21340.00       500.00
1974-07-26  DESIGNER       16  M    1952-06-25     20450.00       400.00
1966-03-03  DESIGNER       16  M    1941-05-29     27740.00       600.00
1968-08-29  DESIGNER       18  F    1948-03-19     29840.00       600.00
1949-08-17  MANAGER        16  M    1925-09-15     40175.00       800.00
1980-06-19  MANAGER        14  M    1956-12-18     26150.00       500.00
1947-05-05  FIELDREP       16  M    1926-05-17     23840.00       500.00
* End of Result ************* 10 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 24. A Query Result from Using a Labeled Duration

In a date or time expression containing two values, only one value can be a labeled-duration expression. The other value must be a date or time expression.

Chapter 3. Manipulating Query Information

Besides specifying the data that you want to see, you can specify the form in which it appears. The following sections describe how you can manipulate the data with arithmetic operators and functions. You can calculate percentages, join character strings, select constants, and put the rows in an order that makes the query more useful to you. Because there are so many ways to manipulate query information, this chapter includes a section about online HELP information.

Selecting Information Calculated from Table Data

The result in "Selecting Rows That Satisfy a Calculated Condition" on page 38 does not show the calculated income. To show calculated values in a result, you can use an expression in the SELECT clause. For example, type the following statement:

```
select empno,lastname,bonus + comm -
from employee -
where bonus + comm > 3000.00
```

The resulting display is shown in Figure 25.

```
EMPNO   LASTNAME           EXPRESSION 1
------  ---------------  --------------
000010  HAAS                    5220.00
000110  LUCCHESI                4620.00
000020  THOMPSON                4100.00
000030  KWAN                    3860.00
000060  STERN                   3180.00
000070  PULASKI                 3593.00
000050  GEYER                   4014.00
* End of Result *** 7 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 25. A Query Result Calculated from Table Data

The column heading for the calculated value is EXPRESSION 1. It cannot be labeled *BONUS* or *COMM* because the values displayed are not the values stored in the BONUS or COMM columns in the EMPLOYEE table. Rather, they are calculated values. A later section describes how you can use the FORMAT command to create a more meaningful column heading.

Interpreting Arithmetic Errors

Errors, such as divide-by-zero or overflow, can occur during the execution of an arithmetic expression in a SELECT statement. If an arithmetic error occurs, the operation is not terminated. Instead, each field that has an arithmetic error is displayed in the resulting table as a row of number signs (#).

For example, a SELECT A/B is performed on the following table:

A	B
1	1
2	0
3	1
4	1

The result is as follows:

```
 A/B
-----
    1
# # #
    3
    4
```

The number signs indicate the row in which an arithmetic error or exception occurs as a result of dividing by zero.

Note: When an error occurs, ISQL message ARI7717I appears and the display is put in HOLDING status. Clear the display. If more than one arithmetic error occurs in the same operation, no additional messages are displayed.

Selecting Information Using Datetime Arithmetic

The arithmetic operations possible between date values or time values are addition and subtraction. For example, to increase the estimated starting date in the PROJECT table by a certain value and display the result, type the following SELECT statement:

```
select prstdate + 30 days -
from project -
where prstdate = '1982-01-01'
```

The result of this query is shown in Figure 26 on page 43.

```
EXPRESSION 1
--------------
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
1982-01-31
* End of Result *** 19 Rows Displayed ***Cost Estimate is 1********************
```

Figure 26. A Query Result from Using Datetime Arithmetic

In the following example, suppose you want to schedule a status meeting one month after the start date of each project in the PROJECT table. You can find the dates by typing the following:

```
select prstdate,date(prstdate + 1 month) -
from project
```

From this query you obtain two columns, the PRSTDATE date and the status meeting date. Or, you can type the following if you only want the dates of the status meetings:

```
select date(prstdate + 1 month) -
from project
```

For detailed rules on the use of datetime arithmetic, see the *DB2 Server for VSE & VM SQL Reference* manual.

EXERCISE 4 (Answers are in Appendix A, Answers to the Exercises, on page 262.)

Create and type a statement for each of the following:

1. Retrieve the employee number, last name, and the salary plus ten percent from the EMPLOYEE table for people who have a bonus of a thousand dollars or greater.
2. Retrieve the project number and activity number for those projects in the PROJ_ACT table whose activity staff is less than 1.5, even with another half staff added.

Selecting Information Using Database Manager Functions

The functions offered by the database manager consist of column functions and scalar functions. Column functions apply the function to a set of values in a column over multiple rows and produce a single result. Scalar functions apply the function to a single value in each row and produce a result for each row.

Defining Column Functions

An SQL column function takes an entire column of data over multiple rows as its argument and produces a single data item that summarizes the column.

The column functions are:

AVG

Calculates the average value of the item to be selected. This function is for numeric columns only (INTEGER, SMALLINT, DECIMAL, and FLOAT). The keyword DISTINCT can be used with AVG to use only distinct (different) values in calculating the average. The function and keyword are written like this:

```
AVG(DISTINCT item)
```

In computing the average, null values are ignored. If all the items specified are null, the average calculated is null. The data type of this column function has the null permitting attribute and is the same as that of its argument except when the argument is SMALLINT, in which case, the resultant data type is INTEGER.

COUNT

This function can be used in one of two ways:

COUNT(*) returns a number indicating how many rows satisfied the search condition (regardless of whether they contain null values).

COUNT(DISTINCT *column_name*) returns a number indicating how many rows containing different values in the column satisfied the search condition. Duplicate values and null values are not counted.

The data type for this column function is INTEGER. The result can be zero, but not null.

MAX

Finds the largest value for the item to be selected. This function can be applied to columns of any type, except long strings. If applied to a CHAR or VARCHAR type column, alphanumeric ordering is used. The number 9 is greater than 8 and so on to 0. The number 0 is greater than the uppercase Z, which is greater than Y, and so on to A. The character A is greater than the lowercase z, which is greater than y, and so on to a. The character a is greater than the special characters.

If the argument is a datetime data type, then the ordering is in chronological order.

If the argument has a data type of VARCHAR or VARGRAPHIC, and the values differ only by trailing blanks, the MAX function considers the values equal.

In finding the maximum, null values are ignored. If all the items specified are null, null is returned. The data type of this column function has the null permitting attribute and is of the same data type as that of the argument.

MIN

This function works the same way that the MAX function works, except that it finds the smallest value of the item to be selected instead of the largest. For details, see the paragraph above describing MAX function.

SUM

Calculates the total value of the item to be selected. This function is for numeric columns only (INTEGER, SMALLINT, DECIMAL, and FLOAT). The keyword DISTINCT can be used with SUM to use only distinct (different) values in calculating the sum. The function and the keyword are written like this:

```
SUM(DISTINCT item)
```

In computing the total, null values are ignored. If all the items specified are null, the total calculated is null. The data type of this column function has the null permitting attribute and is the same as that of its argument except when the argument is SMALLINT type, in which case, the resultant data type is INTEGER.

Using Column Functions

The five column functions, AVG, COUNT, MAX, MIN, and SUM, can be applied to data retrieved from a table.

You apply these functions to a column name or expression in the SELECT clause. They calculate a single value for the column or expression to which they are applied. Therefore, if a column function is applied to one column or expression in a SELECT clause, then all selected columns and expressions must have an associated column function. The one exception to this rule is *grouping*. It is explained under "Selecting Summary Information by Groups" on page 139.

For example, suppose you need to know the average estimated mean number of employees needed to staff an activity, as well as the total of these mean numbers, for project AD3111. Type:

```
select avg(acstaff),sum(acstaff) -
from proj_act -
where projno = 'AD3111'
```

This displays Figure 27.

```
                  AVG(ACSTAFF)                       SUM(ACSTAFF)
---------------------------------  ---------------------------------
     0.93571428571428571428571428571428                             6.55
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 27. A Query Result from Using Column Functions AVG and SUM

As shown in the above example, the item in the SELECT clause that the column function is applied to must be enclosed in parentheses.

The column functions MIN, MAX, and SUM are treated in the same manner as AVG. COUNT can be used to count the number of rows selected (by using an asterisk in place of a column name), or it can be used to count the number of distinct values in a particular column of the rows selected. For example, the following query counts how many rows for employee 130 are contained in the EMP_ACT table:

```
select count(*) -
from emp_act -
where empno = '000130'
```

This produces the display in Figure 28.

```
  COUNT(EXPRESSION 1)
---------------------
                    3
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 28. A Query Result from Using the COUNT Column Function

Notice the column heading for the calculated value is COUNT(EXPRESSION 1). The * in the query is replaced by EXPRESSION 1.

The following query counts how many distinct department numbers are contained in the PROJECT table:

```
select count(distinct deptno) -
from project
```

This presents the display in Figure 29.

```
  COUNT(DISTINCT DEPTNO)
------------------------
                       8
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 29. A Query Result from Using DISTINCT with the COUNT Column Function

Notice that the column heading DISTINCT DEPTNO is not replaced by EXPRESSION 1 because DISTINCT DEPTNO describes the values displayed.

Note: The column functions AVG and SUM cannot be used on columns that contain date, time, or timestamp values.

Defining Scalar Functions

Scalar functions can be used wherever an expression can be used. The function acts on one row of a table at a time and is applied to a single value within the row. The function produces a result for each row.

Scalar functions are:

CHAR
gives character representation of a decimal, timestamp, or datetime value in a specified format.

Suppose the activity start date has an internal representation of `1982-02-15`, and you would like to see the date in USA (character) format (`02/15/1982`), you would type:

```
char(acstdate, usa)
```

DATE

creates a date, as data type DATE, from an expression.

The following example shows how to select all rows from the IN_TRAY table, representing notes that were received on January 1, 1965:

```
select source, subject from in_tray -
where date(received) = '1965-01-01'
```

DAY

extracts the day portion of a date, timestamp, date duration, or timestamp duration.

The following example shows how to select the day part of the column HIREDATE:

```
select day(hiredate) from employee
```

DAYS

extracts the number of days since December 31, 0000.

The following example shows how to obtain the number of days between the HIREDATE and the current date:

```
days(current date) - days(hiredate)
```

DECIMAL(*expression***)**
DECIMAL(*expression***,***precision***)**
DECIMAL(*expression***,***precision***,***scale***)**

returns a decimal representation of a numeric value. The precision and scale of the result you want are specified by the second and third parameters respectively. These two parameters are optional. See the *DB2 Server for VSE & VM SQL Reference* manual for their defaults. In the following example, the precision is 8 and the scale is 2 for one and a half times the salary:

```
select decimal(salary * 1.5,8,2) from employee
```

DIGITS

returns a character representation of a numeric value. The result does not include a sign or decimal point.

The following example shows how to obtain a character representation of EDLEVEL, which is of data type SMALLINT:

```
select digits(edlevel) from employee
```

FLOAT

returns a double-precision, floating-point representation of a numeric value.

The following example shows how to obtain ACTNO in floating-point notation:

```
select float(actno), actdesc from activity
```

HEX

returns a hexadecimal representation of a value.

The following example shows how to convert the activity number in ACTIVITY to a hexadecimal representation:

```
select hex(actno) from activity
```

HOUR

extracts the number of hours from a time, timestamp, time duration, or timestamp duration.

The following example shows how to select rows where the STARTING (a TIME data type) is between 12:00 and 16:00:

```
select * from cl_sched -
where hour(starting) between 12 and 16
```

INTEGER

returns an integer representation of a numeric value. If the value has a decimal part, it is truncated.

The following example shows how you can obtain an integer value in the select list:

```
select integer(salary * 1.5) from employee
```

LENGTH

returns the actual length of an argument.

The following example shows how to find the length of the data in the FIRSTNME field:

```
select length(firstnme) from employee
```

MICROSECOND

extracts the number of microseconds from a timestamp or timestamp duration.

The following example shows how to obtain the microsecond value from the RECEIVED column (a TIMESTAMP data type):

```
select microsecond(received) from in_tray
```

MINUTE

extracts the number of minutes from a time, timestamp, time duration, or timestamp duration. The following example shows how to select all rows from the CL_SCHED table where minutes in ENDING (a TIME data type) is less than 30:

```
select * from cl_sched -
where minute(ending) < 30
```

MONTH

extracts the number of months from a date, timestamp, date duration, or timestamp duration.

The following example shows how to obtain the month portion of the PRSTDATE column:

```
select month(prstdate) from project
```

SECOND

extracts the number of seconds from a time, timestamp, time duration, or timestamp duration.

The following example shows how to obtain the second portion when STARTING (a TIME data type) is subtracted from the CURRENT TIME special register:

```
select second(current time - starting) -
from cl_sched
```

STRIP(*string***)**
STRIP(*string***,***option***)**
STRIP(*string***,***option***,***char***)**
removes blanks and other specified characters from a string. The second parameter indicates the placement of the characters to be stripped (leading, trailing or both), while the third parameter indicates the character to be stripped. These two parameters are optional. See the *DB2 Server for VSE & VM SQL Reference* manual for their defaults.

The following example shows how to strip leading zeros from a number:

```
select strip(mgrno,l,'0') from department
```

MGRNO is the name of the column, `l` is for leading, and `0` is the character to be removed.

SUBSTR(*string***,***start***)**
SUBSTR(*string***,***start***,***length***)**
returns a smaller part (substring) of a string beginning at a *start* position and continuing for a *length*. The length parameter is optional. See the *DB2 Server for VSE & VM SQL Reference* manual for its default.

The following example shows how you can list all projects starting with *A*:

```
select * from project -
where substr(projname,1,1) = 'a'
```

TIME
creates a time from a time, a timestamp, or a string representation of a time.

The following example shows how to obtain a time greater than 5:00 from the RECEIVED column (a TIMESTAMP data type):

```
select received, source, subject from in_tray -
where time(received) > '5.00'
```

This example uses the ISO time format; see “Identifying Datetime Types” on page 179 for other time formats.

TIMESTAMP
creates a timestamp from a date and a time.

The following example shows how to obtain the timestamp value from the CURRENT DATE special register and STARTING (a TIME data type):

```
select timestamp(current date,starting) -
from cl_sched
```

TRANSLATE(*string***,***to_string***)**
TRANSLATE(*string***,***to_string***,***from_string***)**
TRANSLATE(*string***,***to_string***,***from_string***,***pad_character***)**
changes characters in a string into other characters or can be used to reorder characters in a string. When *string* is a character-compatible string, and *to_string* and *from_string* are not supplied, the string is folded to uppercase using the character set specified in SYSCHARSETS. Long field strings are not supported.

The following example shows how to convert the string abbc, replacing every b in the input string to a $ in the output string. Because no conversion character is provided for a or c, they are not converted.

```
translate('abbc','$','b')
```

The string resulting from the conversion is `a$$c`.

The following example shows how to use the TRANSLATE function to reorder the characters in a string. The format of the date is changed from *yyyy-mm-dd* to *dd-mm-yyyy*.

```
select translate('gh-ef-abcd',emendate,'abcd-ef-gh')-
from emp_act
```

VALUE(*expression***,***expression***)**
returns the first non-null result in a series of expressions.

The following example shows how you can set up a default value of zero if the ACTNO is undefined (that is, null):

```
select value(actno,0) from activity
```

VARGRAPHIC
returns a graphic string representation of a character string.

The following example shows how to convert ACTKWD in the ACTIVITY table to graphic characters:

```
select vargraphic(actkwd) from activity
```

YEAR
extracts the number of years from a date, timestamp, date duration, or timestamp duration.

The following example shows how to select all rows where the year portion of ACSTDATE is greater than 1982:

```
select * from proj_act -
where year(acstdate) > 1982
```

For more information on functions, see the *DB2 Server for VSE & VM SQL Reference* manual.

Using Scalar Functions

Scalar functions can be grouped into three general categories. The first category applies to datetime values and can extract or create information; the second category manipulates character string values; the third category converts data from one type to another.

Datetime Functions

The twelve scalar functions that can be applied to date or time values are CHAR, DAY, DAYS, DATE, HOUR, MICROSECOND, MINUTE, MONTH, SECOND, TIME, TIMESTAMP, and YEAR.

You apply these functions to a column name, datetime value, or expression in the SELECT or WHERE clause.

To illustrate the usefulness of these functions, suppose you want to query the PROJECT table to determine which projects have a year or more estimated for the completion of the project. You can use the DAYS function in the query:

```
select projname -
from project -
where days(prstdate) <= days(prendate) - 365
```

The DAYS function serves to turn the date fields into the number of days since December 31, 0000; then you can compare the two dates as numbers, and perform arithmetic operations with them.

Character-String Functions

The three functions that manipulate strings are STRIP, SUBSTR, and TRANSLATE. You apply these functions to a column name, string value, or expression in the SELECT or WHERE clause.

These functions can be used many ways. Use SUBSTR to list all project numbers starting with an `I`:

```
select * -
from proj_act -
where substr(projno,1,1) = 'I'
```

TRANSLATE can be used to change a character string from lowercase characters into uppercase characters. For example, if employee first names were typed in both uppercase and lowercase characters, TRANSLATE can be used as follows to display all names in uppercase characters only:

```
select translate(firstnme) from employee
```

Note: Converting lowercase characters to uppercase characters is useful only if your display is in mixed-case characters.

It may be necessary to change the value of a character in a string for your display. TRANSLATE changes each *S* in the fields of the LASTNAME column in the EMPLOYEE table to a *$* as shown in the following statement:

```
select translate(lastname,'$','S') from employee
```

TRANSLATE can also reorder a string such as the day/month/year sequence in a date. You can convert the date representation in the PRSTDATE column of the PROJECT table from year/month/day to month/day/year as follows:

```
select translate('ab/cd/efgh',prstdate,'efgh/ab/cd') -
from project
```

The first argument in this TRANSLATE function call indicates the desired order for the characters in the string that is the second argument. The third argument represents the current order of the characters in the second argument.

Note: The strings in the previous examples are converted for display only. Use of the TRANSLATE function in a SELECT statement does not affect the data stored in the tables.

Data-Conversion Functions

The six functions that convert data from one data type to another are DECIMAL, DIGITS, FLOAT, HEX, INTEGER, and VARGRAPHIC.

You apply these functions to a column name, a value, or an expression in the SELECT or WHERE clause.

INTEGER can be used to ensure an INTEGER value when calculating values as follows:

```
select integer(prstaff * 1.5) -
from project
```

Ordering Rows of a Query Result

So far, the rows of query results have been shown in an order determined by the database manager. You can also determine the order yourself. For example, suppose you want to select the project number, project name, and estimated mean project staffing value from the PROJECT table for estimated means 1 and 2, and you want the results ordered by estimated mean. Do this by adding an ORDER BY clause to your query like this:

```
select projno,projname,prstaff -
from project -
where prstaff = 1 or prstaff = 2 -
order by prstaff
```

This query displays the result shown in Figure 30.

```
PROJNO  PROJNAME                 PRSTAFF
------  ------------------------ -------
IF2000  USER EDUCATION              1.00
OP2012  APPLICATIONS SUPPORT        1.00
OP2013  DB/DC SUPPORT               1.00
OP2011  SCP SYSTEMS SUPPORT         1.00
PL2100  WELD LINE PLANNING          1.00
AD3112  PERSONNEL PROGRAMMING       1.00
MA2111  W L PROGRAM DESIGN          2.00
IF1000  QUERY SERVICES              2.00
AD3113  ACCOUNT PROGRAMMING         2.00
AD3111  PAYROLL PROGRAMMING         2.00
* End of Result *** 10 Rows Displayed ***Cost Estimate is 1********************
```

Figure 30. A Query Result Where the Rows Are in Order

An ORDER BY clause is always the last clause of a query, and it takes the following form:

```
               ┌─,──────────────────────────┐
                                ┌─ASC──┐
►──ORDER BY───▼─┬─column_name─┬─┼──────┼─┴──────────────────────►
                └─integer─────┘ └─DESC─┘
```

The ORDER BY clause puts the displayed table in order by the values of the columns you identify. The database manager orders the rows in an ascending order unless you indicate that you want them in descending order. If you identify more than one column, the database manager orders the rows by the values of the first column you identify, then by the values of the second column, and so on. For example, to order the above query results primarily by estimated mean and secondarily by project number, type:

```
select projno,projname,prstaff -
from project -
where prstaff = 1 or prstaff = 2 -
order by prstaff,projno
```

This query produces the result in Figure 31.

```
PROJNO  PROJNAME                  PRSTAFF
------  ------------------------  -------
AD3112  PERSONNEL PROGRAMMING        1.00
IF2000  USER EDUCATION               1.00
OP2011  SCP SYSTEMS SUPPORT          1.00
OP2012  APPLICATIONS SUPPORT         1.00
OP2013  DB/DC SUPPORT                1.00
PL2100  WELD LINE PLANNING           1.00
AD3111  PAYROLL PROGRAMMING          2.00
AD3113  ACCOUNT PROGRAMMING          2.00
IF1000  QUERY SERVICES               2.00
MA2111  W L PROGRAM DESIGN           2.00
* End of Result *** 10 Rows Displayed ***Cost Estimate is 1********************
```

Figure 31. A Query Result Ordered According to Two Columns

Trailing blanks in VARCHAR and VARGRAPHIC columns do not affect the order of rows when you use the ORDER BY clause. The database manager removes trailing blanks before it compares VARCHAR or VARGRAPHIC rows, so that two entries that differ only by trailing blanks may not maintain their relative positions.

Defining a field procedure for a column can affect its order when you use the ORDER BY clause. For more information about field procedures, see the *DB2 Server for VSE & VM SQL Reference* manual.

Establishing Field Procedures

The standard EBCDIC sorting sequence may be incorrect or insufficient for some applications. Languages that do not use the Roman alphabet, such as Kanji (Japanese), can only be sorted properly using a field procedure. You can use a field procedure to alter the sorting sequence of values entered in a single short string column (CHAR, VARCHAR, GRAPHIC, VARGRAPHIC); however, you must write a routine to establish the sorting sequence of values.

The database manager creates indexes by using the sorting sequence that the field procedure defines. Column values are sorted using the new sequence. With the CREATE TABLE or ALTER TABLE statement, you can assign a field procedure to a new column by issuing the FIELDPROC clause. However, if you have created a column without a field procedure, you cannot add one later, unless you unload the data and recreate the table to include the field procedure. For further details on writing field procedures, refer to the following manuals:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VM System Administration*

- *DB2 Server for VSE Application Programming*
- *DB2 Server for VM Application Programming*.

Basing the Ordered Rows on a Calculated Result

The order of rows can also be based on a column result that is calculated. For example, suppose you want to know any salaries which exceed $40000 when multiplied by 1.1, and you want the results in descending order by salaries. Do this by typing:

```
select empno,salary * 1.1 -
from employee -
where salary*1.1 > 40000.00 -
order by 2 desc
```

This query displays Figure 32.

```
EMPNO      EXPRESSION 1
------  ---------------
000010        58025.000
000110        51150.000
000020        45375.000
000050        44192.500
000030        42075.000
* End of Result *** 5 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 32. A Query Result Ordered by Column 2 in Descending Order

Here the ORDER BY clause contains a number instead of a column name, and DESC to indicate descending order. The number refers to the position of the item in the SELECT clause to be used for ordering. In the above example, a number was necessary because you were ordering by a calculated column (one that does not exist in the table). For a column that does exist in the table, either its name or a number can be used.

Using the Concatenation Operator

You can join two character strings by using the concatenation operator (CONCAT). This operator is useful if you want to display the character data from two columns without having extra blanks between the entry in each field. For example, to display the first name and last name of each employee in the EMPLOYEE table with only a single space between the first and last name, type:

```
select firstnme CONCAT ' ' CONCAT lastname -
from employee
```

FIRSTNME and LASTNAME are variable-length fields. Only the actual length of the string in the field is concatenated for the display. Fixed-length strings retain the extra blanks even when concatenated. To limit the number of blanks when concatenating fixed-length strings, you can use the STRIP function to remove trailing blanks and then concatenate the results.

The concatenation operator can also add string constants such as commas, dashes, or words to the display. For example, to separate the employee number and the employee's first name with a dash, type:

```
select strip(empno) CONCAT ' - ' CONCAT firstnme -
from employee
```

The above SELECT statement uses the STRIP scalar function to remove trailing blanks from each employee number (a fixed-length string) before concatenating the number, a dash, and the first name.

The concatenation operator joins any two strings as long as they are compatible data types: two character strings, a character string and a datetime value, or two graphic strings. You can also use || as a synonym for CONCAT.

Selecting Constants

A constant can be a numeric value, hexadecimal data, or character data. Constants that contain character or hexadecimal data must be enclosed in *single* quotation marks. To represent a single quotation mark within a constant, use two single quotation marks. For example, to represent the constant BILL'S HOTEL, use:

```
'bill''s hotel'
```

Hexadecimal data can be used for both character and DBCS data representation. To represent the word BOAT with a hexadecimal constant, use the following:

```
X'C2D6C1E3'
```

To represent graphic constants, you must use shift-out and shift-in characters as follows:

```
G'so...si'
```

where `so` is the shift-out character (X'0E'), `si` is the shift-in character (X'0F'), and `...` is graphic data (a DBCS string).

In addition to selecting data from a table, the SELECT statement can be used to create a column containing some desirable constant. For example, write the project number for the query shown on page 45 by including the project number as a constant as follows:

```
select avg(acstaff),sum(acstaff),'AD3111' -
from proj_act -
where projno = 'AD3111'
```

This displays the result in Figure 33.

```
                     AVG(ACSTAFF)                       SUM(ACSTAFF)  EXPRESSIO
 --------------------------------  --------------------------------  ---------
      0.935714285714285714285714                                6.55  AD3111
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*******************
```

Figure 33. A Query Result with the Project Number as a Constant

On queries that return multiple rows, the constant appears in every row of the result.

EXERCISE 5 (Answers are in Appendix A, Answers to the Exercises, on page 262.)

Create and type a statement for each of the following:

1. Retrieve the total mean number of staff needed for activity number 20 if the mean number of staff was reduced to three quarters of its present size.
2. Retrieve the count of how many different departments are listed in the PROJECT table.
3. Retrieve the minimum salary for department D11 from the EMPLOYEE table.

Obtaining Online HELP Information at a Terminal

The database manager offers online HELP information that provides you with, for example, information about ISQL commands; this saves you time in looking up the commands in Chapter 13, "ISQL Commands" on page 203.

Online HELP information is available for:

- Reference information
- SQL statements
- ISQL commands
- Messages, codes and SQLSTATEs.

Online HELP information is not supported on non-DB2 Server for VSE & VM application servers.

Selecting Online HELP Information

Online HELP information may also have been installed on your system in languages other than English. If you want to read HELP information in one of these other languages, you can specify the language for online HELP using the SET LANGUAGE command, which is discussed in "Language of Messages and HELP Text" on page 115 and in Chapter 13, "ISQL Commands" on page 203.

The database manager online HELP information is stored in a table and displayed in the same manner as queries. After you have retrieved some information, you can use display commands to move through it. The online information is broken up into many topics. Each topic is identified by a name. Most topic names are either a statement name (such as SELECT or INSERT), a message number (such as ARI7399I or ARI7307A), a message code (such as -205 or 100), or an SQLSTATE (such as SQLSTATE 01525). For example, to retrieve the online help information for the UPDATE statement, type:

```
help update
```

To move forward through the UPDATE information, type FORWARD commands until you have reached the end. See "Results That Have Too Many Rows for One Display" on page 62 for more information about FORWARD. Type the END command to end the display.

Some topic names have more than one word. When retrieving these topics, enclose the topic name in single quotation marks. For example, to retrieve the online help information for search conditions, type:

```
help 'search conditions'
```

To see a list of the topics available, type:

```
help contents
```

To obtain a list of topics, and a description of how to use the HELP command, press PF1 or type:

```
help
```

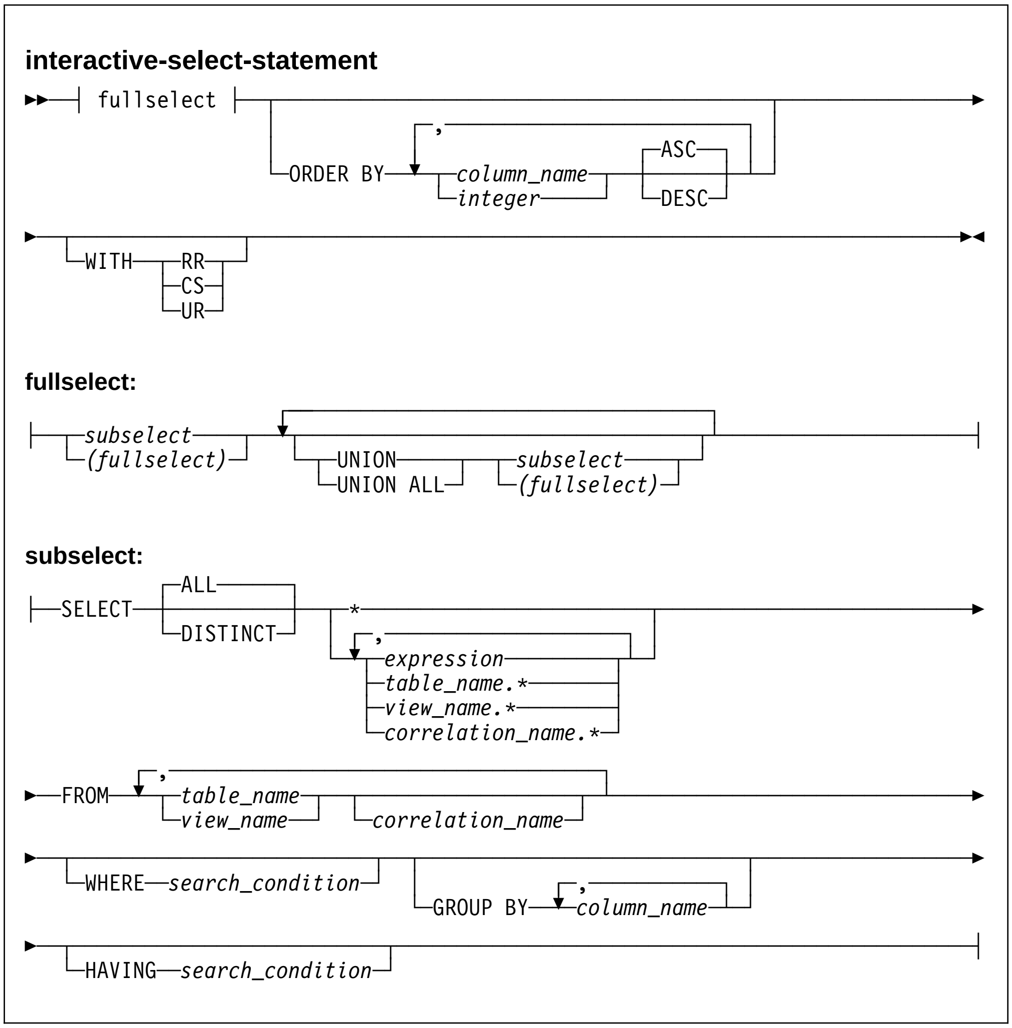
Typing While Viewing Online HELP Information

You can type commands and statements while online HELP information is being displayed. This is particularly useful when you want to type a statement, and you know all the necessary column and table name information, but you cannot remember the correct format for the statement.

If you want to execute a SELECT statement but cannot remember its format, type:

```
help select
```

This displays online HELP information for the SELECT statement similar to the following:




```
PROJNO  PRSTAFF
------  -------
IF2000     1.00
OP2012     1.00
OP2013     1.00
OP2011     1.00
PL2100     1.00
AD3112     1.00
MA2111     2.00
IF1000     2.00
AD3113     2.00
AD3111     2.00
* End of Result *** 10 Rows Displayed ***Cost Estimate is 1********************
```

Figure 34. A Query Result after Using Online HELP Information

EXERCISE 6 (Answers are in Appendix A, Answers to the Exercises, on page 262.)

Create and type a statement for each of the following:

1. Retrieve the rows of the PROJ_ACT table that pertain to project number OP2010. Order the results primarily by activity staff and secondarily by activity number.
2. Retrieve the project numbers and project names from the PROJECT table that end on February 1, 1983, and also use the constant *ends Feb 1, 1983* in the results. Order the results by project number in descending order.

Chapter 4. Using Query Results

Previous chapters described how to display the query result that you want. Now that you have it on your display, you can look at it, and perhaps print it. The following sections show how to look at the entire query result if it does not all fit on your display, and how to direct the query result to a printer.

Displaying Query Results

So far, you have used only one of the display commands that can be entered while displaying query results: the END command (PF3). Because the sample tables are small, END is generally the only display command you require; however, when your query results are larger than a single display, you must know the display commands with which you can view different portions of the query result. The following examples show how to use the other display commands to query longer and wider tables.

Type the following query and press ENTER:

```
select * -
from proj_act -
where projno <> 'AD3111' -
order by actno,acendate
```

This query selects all the rows that contain project numbers other than *AD3111* in the column named PROJNO from the PROJ_ACT table. It produces a result similar to the display in Figure 35.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3100      10     0.50  1982-01-01  1982-07-01
MA2100      10     0.50  1982-01-01  1982-11-01
IF2000      10     0.50  1982-01-01  1983-01-01
IF1000      10     0.50  1982-06-01  1983-01-01
IF1000      10     0.50  1982-01-01  1983-01-01
AD3110      10     1.00  1982-01-01  1983-01-01
OP1000      10     0.25  1982-01-01  1983-02-01
OP1010      10     1.00  1982-01-01  1983-02-01
OP2010      10     1.00  1982-01-01  1983-02-01
MA2110      10     1.00  1982-01-01  1983-02-01
MA2100      20     1.00  1982-01-01  1982-03-01
PL2100      30     1.00  1982-02-01  1982-09-01
PL2100      30     1.00  1982-01-01  1982-09-15
MA2111      40     1.00  1982-01-01  1983-02-01
MA2111      50     1.00  1982-01-01  1982-06-01
OP2000      50     0.75  1982-01-01  1983-02-01
AD3112      60     0.75  1982-01-01  1982-03-15
AD3112      60     0.50  1982-02-01  1982-03-15
MA2112      60     2.00  1982-01-01  1982-07-01
AD3113      60     1.00  1982-04-01  1982-09-01
MA2113      60     1.00  1982-02-15  1982-09-01
AD3113      60     0.75  1982-03-01  1982-10-15
```

Figure 35. Example of a Long Query Result

This query result is used in the following topics. Do not end it yet.

Results That Have Too Many Rows for One Display

The query results that you are retrieving are longer than can be shown on one display. To see the remaining portions of the query result, you must know how to scroll forward through it. Usually, the command to scroll forward 20 rows is FORWARD 20.

Instead of entering FORWARD 20, take advantage of the special function of the ENTER key on query results. While viewing a query result, pressing ENTER without having a command in the input area repeats the previous display command. If there was no previous display command entered, the length of the display is scrolled forward. If the previous command is BACKWARD MAX and the display starts from the beginning of the table, press ENTER to scroll forward the length of the display.

Press ENTER. On a 24 X 80 terminal you see a result similar to the display in Figure 36.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3112      70     0.50  1982-02-01  1982-03-15
AD3113      70     0.50  1982-06-15  1982-07-01
AD3112      70     1.00  1982-03-15  1982-08-15
MA2112      70     1.00  1982-02-01  1982-10-01
AD3112      70     0.75  1982-01-01  1982-10-15
AD3112      70     0.25  1982-08-15  1982-10-15
AD3113      70     0.75  1982-09-01  1982-10-15
AD3113      70     1.25  1982-06-01  1982-12-15
MA2112      70     1.50  1982-02-15  1983-02-01
AD3113      70     1.00  1982-07-01  1983-02-01
AD3113      70     1.00  1982-10-15  1983-02-01
MA2112      70     1.00  1982-06-01  1983-02-01
MA2113      70     2.00  1982-04-01  1983-12-15
AD3113      80     1.75  1982-01-01  1982-04-15
AD3113      80     0.50  1982-03-01  1982-04-15
AD3112      80     0.50  1982-10-15  1982-12-01
AD3112      80     0.35  1982-08-15  1982-12-01
MA2113      80     0.50  1982-10-01  1983-02-01
MA2113      80     1.50  1982-09-01  1983-02-01
MA2113      80     1.00  1982-01-01  1983-02-01
MA2112      80     1.00  1982-10-01  1983-10-01
IF1000      90     0.50  1982-10-01  1983-01-01
```

Figure 36. Display 20 Rows Forward by Pressing ENTER

To scroll through the query result a half display at a time, you can either enter FORWARD or press PF8.

Press PF8.

The rows on the display scroll half the display. The middle row of the previous display is now the first row of the new display.

Now, to move forward to the remaining rows, enter:

```
forward max
```

This results in a display similar to Figure 37 on page 63.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
IF1000      90     0.50  1982-10-01  1983-01-01
IF1000      90     1.00  1982-01-01  1983-01-01
IF2000     100     0.50  1982-03-01  1982-07-01
IF2000     100     0.75  1982-01-01  1982-07-01
IF1000     100     0.50  1982-10-01  1983-01-01
IF2000     110     0.50  1982-03-01  1982-07-01
IF2000     110     0.50  1982-10-01  1983-01-01
OP1010     130     4.00  1982-01-01  1983-02-01
OP2012     140     0.25  1982-01-01  1983-02-01
OP2011     140     0.75  1982-01-01  1983-02-01
OP2013     140     0.50  1982-01-01  1983-02-01
OP2011     150     0.25  1982-01-01  1983-02-01
OP2012     160     0.75  1982-01-01  1983-02-01
OP2013     170     0.50  1982-01-01  1983-02-01
AD3113     180     1.00  1982-04-15  1982-06-01
AD3113     180     0.50  1982-06-01  1982-07-01
AD3113     180     0.75  1982-03-01  1982-07-01
AD3112     180     0.50  1982-08-15  1983-01-01
MA2113     180     0.50  1982-10-01  1983-01-01
MA2112     180     1.00  1982-07-01  1983-02-01
MA2112     180     1.00  1982-07-15  1983-02-01
* End of Result *** 70 Rows Displayed ***Cost Estimate is 1*******************
```

Figure 37. Display after Moving to the End of the Query Result

To move back through the query result, use a BACKWARD command. Also, you can use PF7 to move your view of the result backward one-half display. Moving back through the query result is limited; you can move back only to a limit of one full display from the last FORWARD command. If you want to go farther back, you must return directly to the beginning of the query result.

For example, to view the previous 15 rows, enter:

backward 15

This command presents the display in Figure 38 on page 64.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
AD3112      70     0.50  1982-02-01  1982-03-15
AD3113      70     0.50  1982-06-15  1982-07-01
AD3112      70     1.00  1982-03-15  1982-08-15
MA2112      70     1.00  1982-02-01  1982-10-01
AD3112      70     0.75  1982-01-01  1982-10-15
AD3112      70     0.25  1982-08-15  1982-10-15
AD3113      70     0.75  1982-09-01  1982-10-15
AD3113      70     1.25  1982-06-01  1982-12-15
MA2112      70     1.50  1982-02-15  1983-02-01
AD3113      70     1.00  1982-07-01  1983-02-01
AD3113      70     1.00  1982-10-15  1983-02-01
MA2112      70     1.00  1982-06-01  1983-02-01
MA2113      70     2.00  1982-04-01  1983-12-15
AD3113      80     1.75  1982-01-01  1982-04-15
AD3113      80     0.50  1982-03-01  1982-04-15
AD3112      80     0.50  1982-10-15  1982-12-01
AD3112      80     0.35  1982-08-15  1982-12-01
MA2113      80     0.50  1982-10-01  1983-02-01
MA2113      80     1.50  1982-09-01  1983-02-01
MA2113      80     1.00  1982-01-01  1983-02-01
MA2112      80     1.00  1982-10-01  1983-10-01
IF1000      90     0.50  1982-10-01  1983-01-01
```

Figure 38. Results When You Try to Exceed the Limit of a Full Display

The view is moved back to the limit of one full display. Enter the following command to return to the first rows of the query result:

```
backward max
```

Now that you are finished with this query result, end it.

Results That Are Too Wide for One Display

To learn about viewing results that are too wide for a single display, it is necessary to use a sample table of greater width than the PROJ_ACT table. For the next several examples, the EMPLOYEE table is used.

Enter the following query:

```
select * -
from employee
```

The resulting query on an 80-character display is similar to Figure 39 on page 65.

```
EMPNO   FIRSTNME     MIDINIT  LASTNAME         WORKDEPT  PHONENO  HIREDATE
-----   ------------ -------  ---------------  --------  -------  ----------
000010  CHRISTINE    I        HAAS             A00       3978     1991-05-27
000110  VINCENZO     G        LUCCHESI         A00       3490     1958-05-16
000120  SEAN                  O'CONNELL        A00       2167     1963-12-05
000020  MICHAEL      L        THOMPSON         B01       3476     1973-10-10
000030  SALLY        A        KWAN             C01       4738     1975-04-05
000130  DOLORES      M        QUINTANA         C01       4578     1971-07-28
000140  HEATHER      A        NICHOLLS         C01       1793     1976-12-15
000060  IRVING       F        STERN            D11       6423     1973-09-14
000150  BRUCE                 ADAMSON          D11       4510     1972-02-12
000160  ELIZABETH    R        PIANKA           D11       3782     1977-10-11
000170  MASATOSHI    J        YOSHIMURA        D11       2890     1978-09-15
000180  MARILYN      S        SCOUTTEN         D11       1682     1973-07-07
000190  JAMES        H        WALKER           D11       2986     1974-07-26
000200  DAVID                 BROWN            D11       4501     1966-03-03
000210  WILLIAM      T        JONES            D11       0942     1979-04-11
000220  JENNIFER     K        LUTZ             D11       0672     1968-08-29
000070  EVA          D        PULASKI          D21       7831     1980-09-30
000230  JAMES        J        JEFFERSON        D21       2094     1966-11-21
000240  SALVATORE    M        MARINO           D21       3780     1979-12-05
000250  DANIEL       S        SMITH            D21       0961     1969-10-30
000260  SYBIL        P        JOHNSON          D21       8953     1975-09-11
000270  MARIA        L        PEREZ            D21       9001     1980-09-30
```

Figure 39. Results of Query That is Too Wide for Display

There is no visual indication that this query result is too wide for the display; however, because the columns extend to the far right side of the display, there is a possibility that additional columns of data may be beyond the last column. To search for any additional columns that may exist, enter the following display command:

right 7

This moves your view of the query seven columns to the right, resulting in Figure 40 on page 66.

```
JOB      EDLEVEL  SEX  BIRTHDATE       SALARY        BONUS         COMM
-------- -------  ---  ----------  -----------  -----------  -----------
PRES          18  F    1933-08-01     52750.00      1000.00      4220.00
SALESREP      19  M    1929-11-05     46500.00       800.00      3720.00
CLERK         14  M    1942-10-18     29250.00       600.00      2340.00
MANAGER       18  M    1948-02-02     41250.00       800.00      3300.00
MANAGER       20  F    1941-05-11     38250.00       800.00      3060.00
ANALYST       16  F    1925-09-15     23800.00       800.00      1904.00
ANALYST       18  F    1946-01-19     28420.00       800.00      2274.00
MANAGER       16  M    1945-07-07     32250.00       600.00      2580.00
DESIGNER      16  M    1947-05-17     25280.00       500.00      2022.00
DESIGNER      17  F    1955-04-12     22250.00       400.00      1780.00
DESIGNER      16  M    1951-01-05     24680.00       500.00      1974.00
DESIGNER      17  F    1949-02-21     21340.00       500.00      1707.00
DESIGNER      16  M    1952-06-25     20450.00       400.00      1636.00
DESIGNER      16  M    1941-05-29     27740.00       600.00      2217.00
DESIGNER      17  M    1953-02-23     18270.00       400.00      1462.00
DESIGNER      18  F    1948-03-19     29840.00       600.00      2387.00
MANAGER       16  F    1953-05-26     36170.00       700.00      2893.00
CLERK         14  M    1935-05-30     22180.00       400.00      1774.00
CLERK         17  M    1954-03-31     28760.00       600.00      2301.00
CLERK         15  M    1939-11-12     19180.00       400.00      1534.00
CLERK         16  F    1936-10-05      1725.00       300.00      1380.00
CLERK         15  F    1953-05-26     27380.00       500.00      2190.00
```

Figure 40. Display of Query Moved Seven Columns to the Right

Now you can see that there are an additional seven columns, but it is still not clear that you have reached the last column of the table. Enter the following command to move your view two more columns to the right:

right 2

You should now see the display in Figure 41.

```
SEX  BIRTHDATE       SALARY        BONUS         COMM
---  ----------  -----------  -----------  -----------
F    1933-08-01     52750.00      1000.00      4220.00
M    1929-11-05     46500.00       800.00      3720.00
M    1942-10-18     29250.00       600.00      2340.00
M    1948-02-02     41250.00       800.00      3300.00
F    1941-05-11     38250.00       800.00      3060.00
F    1925-09-15     23800.00       800.00      1904.00
F    1946-01-19     28420.00       800.00      2274.00
M    1945-07-07     32250.00       600.00      2580.00
M    1947-05-17     25280.00       500.00      2022.00
F    1955-04-12     22250.00       400.00      1780.00
M    1951-01-05     24680.00       500.00      1974.00
F    1949-02-21     21340.00       500.00      1707.00
M    1952-06-25     20450.00       400.00      1636.00
M    1941-05-29     27740.00       600.00      2217.00
M    1953-02-23     18270.00       400.00      1462.00
F    1948-03-19     29840.00       600.00      2387.00
F    1953-05-26     36170.00       700.00      2893.00
M    1935-05-30     22180.00       400.00      1774.00
M    1954-03-31     28760.00       600.00      2301.00
M    1939-11-12     19180.00       400.00      1534.00
F    1936-10-05      1725.00       300.00      1380.00
F    1953-05-26     27380.00       500.00      2190.00
```

Figure 41. Display of Query Moved Two More Columns to the Right

Now it is apparent that COMM is the final column in the table, because no other columns appear to the right of it in the query result. If you only want to move your view of the table one column to the right, you can omit the number and just type `right`. You can also move your view one column to the right by pressing PF11.

To move your view to the left, type:

```
left
```

This command works in the same manner as RIGHT. Again, you can move your view one column to the left by pressing PF10.

Now move your view all the way back to the first column by using the following command:

```
column 1
```

The COLUMN command aligns the column you specify with the left edge of the display. The number refers to the column's position in the query result. If you do not specify a number with the COLUMN command, column 1 is placed at the left edge of the display.

A column of characters that is wider than the display width requires another display command, TAB, to let you see the entire length attribute of a column. The TAB command is described in Chapter 13, "ISQL Commands" on page 203.

Obtaining a Printed Report

So far you have learned how to view query results at your display terminal. Now produce a printed copy of a query result on the system printer. The information requested to be printed is a copy of the DEPARTMENT table. Begin by entering the following query:

```
select * -
from department -
order by deptno
```

As you can see, this query selects all the columns and all the rows from the DEPARTMENT table. Request a printed copy of this query result by entering the following display command:

```
print
```

Another way to request a printed copy of the query result is to press PF4. Figure 42 on page 68 illustrates the printed information for the preceding query. Consult the appropriate person at your location for printing procedures.

Each printed page is numbered, dated, and titled. The title consists of the first 100 characters of the SELECT statement that you typed. For information on the creation of report titles, refer to Chapter 7, "Formatting Query Results" on page 95.

```
.     07/13/89     SELECT * FROM DEPARTMENT ORDER BY DEPTNO     PAGE   1 .
.                                                                        .
.                                                                        .
.     DEPTNO  DEPTNAME                     MGRNO   ADMRDEPT              .
.     ______  ____________________________ ______  ________              .
.     A00     SPIFFY COMPUTER SERVICE DIV. 000010  A00                   .
.     B01     PLANNING                     000020  A00                   .
.     C01     INFORMATION CENTER           000030  A00                   .
.     D01     DEVELOPMENT CENTER           ?       A00                   .
.     D11     MANUFACTURING SYSTEMS        000060  D01                   .
.     D21     ADMINISTRATION SYSTEMS       000070  D01                   .
.     E01     SUPPORT SERVICES             000050  A00                   .
.     E11     OPERATIONS                   000090  E01                   .
.     E21     SOFTWARE SUPPORT             000100  E01                   .
.                                                                        .
.                                                                        .
.                                                                        .
.                                                                        .
.                                                                        .
```

Figure 42. Example of a Printed Report

Although this query fits on one display, a long query that requires several displays is also printed entirely, regardless of the rows being displayed when you typed the PRINT command. In addition, when the PRINT command completes printing the report, the end rows of the query result are displayed regardless of the rows being displayed when you entered the PRINT command.

The printed report does, however, start with the column that is currently at the left edge of the display, and continues for as many print positions as can fit on the specified page size. You can print reports that are too wide for the page size by entering a PRINT command at column 1, moving the display to start with the column that would not fit on the page, entering another PRINT command, and so on. Page-size specification is discussed in “Page Size of Printed Reports” on page 114.

You may have special printing requirements for your query results (such as the type of paper or its size) that you can specify using the CLASS keyword of the PRINT command. See the PRINT command in Chapter 13, “ISQL Commands” on page 203 for a description of this procedure. For DB2 Server for VM, the CLASS value can also be set by a CP SPOOL command.

Obtaining Multiple Copies of a Printed Report

For three copies, type:

```
print copies 3
```

DB2 Server for VM

You can also use the CP SPOOL command to specify the number of copies. The value for copies that is specified on the CP SPOOL command remains in effect until another value is specified by another CP SPOOL command.

Note: If you specify the number of copies on the PRINT command after you specified it using a CP SPOOL command, the PRINT command quantity is used for that print operation. All following PRINT commands use the quantity specified by the CP SPOOL command, unless you specify it using the **COPIES** keyword.

Using More Than One Keyword with the Print Command

You can specify both your print requirements and number of copies with one PRINT command. For example, you can type:

```
print copies 3 class a
```

EXERCISE 7 (Answers are in Appendix A, Answers to the Exercises, on page 262.)

Enter the following command:

```
select * -
from proj_act -
where projno <> 'AD3111' -
order by actno,acendate
```

Perform the following:

1. Move the display so that it begins with the 40th row of the query result.
2. Display the end of the query result.
3. Move the display so that it begins with the second column of the query result.
4. Move the display so that it begins with the first column of the query result.
5. Move the display so that it begins with the first row of the query result.
6. Request a printed copy of this result.
7. End the query result.

Chapter 5. Managing Table Data

This chapter shows how to maintain the data in your tables by updating, deleting, and inserting the necessary information. When the tables are interrelated, you must maintain the accuracy of all the tables whenever you make changes. The following sections describe how to control your changes and preserve the relationships between tables.

Controlling Changes to Table Data

Sometimes you want to make updates only with other updates (or deletions). For example, a new employee is hired and is assigned to a project. This involves updates to the EMPLOYEE and EMP_ACT tables. In addition, the employee is made responsible for the project, requiring an update of the PROJECT table (RESPEMP column). The three tables are updated simultaneously, because if only some of the updates are made, and an interval of time passes before the remaining updates are made, anyone accessing the tables during the interval for information about the new employee receives inaccurate information.

For simultaneous updating of tables, you can group SQL statements into a single unit. The statements are run as a group, and only upon your specification. You can also cancel the statements as a group. Such a grouping of one or more statements is called a *logical unit of work* (*LUW*). Two system settings are used in LUW processing: AUTOCOMMIT ON and AUTOCOMMIT OFF.

Using the AUTOCOMMIT ON Setting

With the AUTOCOMMIT ON setting, each statement is treated as an LUW. The work performed by each statement is committed as part of statement processing, resulting in a permanent change. There are, however, a few exceptions. When an INSERT, UPDATE, or DELETE statement that affects more than one row is typed, the LUW is not completed until the next statement is typed. If the next statement is CANCEL or ROLLBACK, the work performed by the INSERT, UPDATE, or DELETE operation before the CANCEL or ROLLBACK is undone instead of being committed. If the next statement is not CANCEL or ROLLBACK, the work performed by the INSERT, UPDATE, or DELETE is committed automatically before the new statement is processed. The CANCEL command is described under "Cancelling Running Commands" on page 171 and Chapter 13, "ISQL Commands" on page 203.

The default setting in ISQL is AUTOCOMMIT ON.

Using the AUTOCOMMIT OFF Setting

Use the AUTOCOMMIT OFF setting to control the committing of information. After you type SET AUTOCOMMIT OFF, any subsequent statements you type are grouped into a single LUW. The statements are processed but not committed until you type the SQL statement COMMIT. As you type the statements, the table data shown on the display looks as if the changes are being committed: they are not. In addition, no other system user can view or modify the changes that you are making until you type the COMMIT statement. Typing COMMIT completes your LUW and commits all processing performed since the beginning of the LUW.

After you set AUTOCOMMIT OFF, and if for any reason, such as an update error, you decide not to commit the LUW processing, you can cancel all processing performed since the beginning of the LUW by typing ROLLBACK. After you set AUTOCOMMIT OFF, you must explicitly end the LUW by typing COMMIT or ROLLBACK.

Attention Use AUTOCOMMIT OFF cautiously because it prevents other users from accessing the rows of tables used in your processing. In particular, avoid using it for a query.

To return to the condition in which ISQL automatically commits your work, type `SET AUTOCOMMIT ON`. The system displays a message requesting you to commit or rollback any work done during the logical unit of work. You must respond to this message before any further statements can be typed.

Interpreting Messages While Making Changes

Messages are displayed after data has been inserted, deleted, or updated to indicate the number of rows affected by each operation. These messages help you verify the changes.

Interpreting Errors While Making Changes

A single statement can change many rows in a table. If a statement error occurs after only some rows have been changed, all changes are rolled back or withdrawn, and the entire operation fails. If the failed statement is part of an LUW, previously completed statements in the LUW are *not* affected. You still have the option of entering COMMIT or ROLLBACK for the other statements in the LUW.

Understanding Referential Integrity

The database manager provides you with *referential integrity* to maintain the integrity of the data in your tables. The following paragraphs discuss important aspects of referential integrity.

Defining Referential Integrity

Referential integrity is the condition in which the existence of values in one table depends on the existence of the same values in another table. For example, you want to ensure that only valid, existing department numbers from your DEPARTMENT table are stored in your EMPLOYEE table. You may also want any deletions or changes to department numbers in the DEPARTMENT table to be prevented until the changes are reflected in the corresponding rows in the EMPLOYEE table. The database manager maintains referential integrity through the use of table keys and a set of rules known as *referential constraints* that define insert, update, and delete rules for tables in referential relationships.

Defining a Primary Key

In some tables, each row represents a unique item. For example, each row in the EMPLOYEE table in Figure 43 on page 73 represents a different employee.

EMPNO	FIRSTNAME	LASTNAME	WORKDEPT . . .
00010	CHRISTINE	HAAS	A00
00110	VINCENZO	LUCCHESI	A01
00120	SEAN	O'CONNELL	A01
00020	MICHAEL	THOMPSON	B01
00030	SALLY	KWAN	C01
.	.	.	.
.	.	.	.
.	.	.	.

Figure 43. EMPLOYEE Table

Each employee is identified by a unique employee number in the EMPNO column, which has been defined as the *primary key*.

A primary key is a column or a set of columns that uniquely identifies each row. The primary key is part of the table definition. When a column or columns is defined as a primary key, the database manager monitors all changes to the table, preventing the insertion of null or duplicate values in the primary key column.

Defining a Foreign Key

Other tables can access data in a table with a primary key, and the system checks that references between the tables are valid. In the sample application illustrated in Figure 44 on page 74, the EMPLOYEE table serves as a master list of all employees, the DEPARTMENT table acts as a master list of all valid departments, and the PROJECT table provides a master list of current projects.

Other tables refer to employees, departments, or projects by using the unique values for each row in the master lists. These matching values are contained in a column or set of columns that have the same data types and attributes as the primary key in the master list. The column (or set of columns) in one table that refers to a primary key in another table is a *foreign key*. The foreign key must have the same number of columns as the primary key that it references. The maximum number of columns that can form a primary or foreign key is 16.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-14	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	600	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

Figure 44. Relationships Among Tables in the Sample Application

Foreign key values, such as employee number in the EMP_ACT table, are controlled to make sure that each foreign key value references a valid row in the master list. Each foreign key value must match a primary key value or else be null. A foreign key value in a row is null when any one of the fields that make up the foreign key value in that row is null. Primary keys cannot consist of columns containing null values. A foreign key value that matches a primary key value is *dependent* on that primary key value.

Establishing a Referential Constraint

The relationship between the primary key and the foreign key is formalized in a referential constraint. A referential constraint identifies the table in which the primary key occurs and the foreign key referencing it. Each referential constraint involves a set of rules to ensure that each foreign key value matches a primary key value (or is null) regardless of updates, insertions, and deletions in either the *parent table* or the *dependent table*. The parent table contains the primary key; the dependent table contains the foreign key.

A referential constraint is part of the dependent table's definition. After a table has been set up as part of a *referential structure*, subsequent access to both parent and dependent tables is controlled.

As illustrated in Figure 44, tables can have only one primary key, For example, EMPLOYEE is the parent table and DEPARTMENT is the dependent table in the referential constraint involving the foreign key MGRNO (DEPARTMENT) and the primary key EMPNO (EMPLOYEE). EMPLOYEE is the dependent table and

DEPARTMENT is the parent table in the referential constraint involving the foreign key WORKDEPT (EMPLOYEE) and the primary key DEPTNO (DEPARTMENT).

The referential structure in this manual involves only six tables. But even in so simple an application, it is obvious how changes to rows in one table can affect the validity of rows in another. For example, changes to a department number in the DEPARTMENT table can affect rows in the PROJECT or EMPLOYEE tables. Changes to an employee row can affect the DEPARTMENT, PROJECT, and the EMP_ACT tables.

Users who access tables in a referential structure such as this, however, need not set up any constraints themselves. The database manager implements the constraints.

For more information on referential integrity, see the *DB2 Server for VSE Database Administration* and *DB2 Server for VM Database Administration* manuals.

Updating Table Data

Updating Rows

If you have not entered SET AUTOCOMMIT OFF, do so now. This ensures that any updates to the table data can be rolled back if necessary. For more information about the AUTOCOMMIT OFF setting, see “Using the AUTOCOMMIT OFF Setting” on page 71.

Suppose you want to advance the completion date by 5 days for activity 100 in project IF1000, which is updating the appropriate EMENDATE column in the EMP_ACT table, and the ACENDATE column in the PROJ_ACT table. Assume that you queried the tables and know that both columns contain the value 1983-01-01. To update the EMP_ACT table, type:

```
update emp_act -
set emendate = '1983-01-06' -
where projno = 'IF1000' -
and actno = 100
```

After this statement is entered, you receive a message indicating that one row is updated.

To update the PROJ_ACT table, type:

```
update proj_act -
set acendate = '1983-01-06' -
where projno = 'IF1000' -
and actno = 100
```

In the previous update statements, you supplied the new date value. As an alternative, the database manager can calculate the new value for you by specifying that 5 days be added to the current value. You can specify the following in the SET clause:

```
set acendate = acendate + 5 days
```

Query the EMP_ACT table to verify the change. The display is similar to Figure 45 on page 76.

```
EMPNO   PROJNO   ACTNO  EMPTIME  EMSTDATE    EMENDATE
------  ------  ------  -------  ----------  ----------
000030  IF1000      10     0.50  1982-06-01  1983-01-01
000130  IF1000      90     1.00  1982-01-01  1982-10-01
000130  IF1000     100     0.50  1982-10-01  1983-01-06
000140  IF1000      90     0.50  1982-10-01  1983-01-01
000030  IF2000      10     0.50  1982-01-01  1983-01-01
000140  IF2000     100     1.00  1982-01-01  1982-03-01
000140  IF2000     100     0.50  1982-03-01  1982-07-01
000140  IF2000     110     0.50  1982-03-01  1982-07-01
000140  IF2000     110     0.50  1982-10-01  1983-01-01
000010  MA2100      10     0.50  1982-01-01  1982-11-01
000110  MA2100      20     1.00  1982-01-01  1982-03-01
000010  MA2110      10     1.00  1982-01-01  1983-02-01
000200  MA2111      50     1.00  1982-01-01  1982-06-15
000200  MA2111      60     1.00  1982-06-15  1983-02-01
000220  MA2111      40     1.00  1982-01-01  1983-02-01
000150  MA2112      60     1.00  1982-01-01  1982-07-15
000150  MA2112     180     1.00  1982-07-15  1983-02-01
000170  MA2112      60     1.00  1982-01-01  1983-06-01
000170  MA2112      70     1.00  1982-06-01  1983-02-01
000190  MA2112      70     1.00  1982-02-01  1982-10-01
000190  MA2112      80     1.00  1982-10-01  1983-10-01
000160  MA2112      60     1.00  1982-07-15  1983-02-01
```

Figure 45. Verify Changed Dates in EMP_ACT Table

Now query the PROJ_ACT table. It resembles Figure 46.

```
PROJNO   ACTNO  ACSTAFF  ACSTDATE    ACENDATE
------  ------  -------  ----------  ----------
IF1000      10     0.50  1982-01-01  1983-01-01
IF1000      10     0.50  1982-06-01  1983-01-01
IF1000      90     1.00  1982-01-01  1983-01-01
IF1000      90     0.50  1982-10-01  1983-01-01
IF1000     100     0.50  1982-01-01  1983-01-06
IF2000      10     0.50  1982-01-01  1983-01-01
IF2000     100     0.75  1982-01-01  1982-07-01
IF2000     100     0.50  1982-03-01  1982-07-01
IF2000     110     0.50  1982-03-01  1982-07-01
IF2000     110     0.50  1982-10-01  1983-01-01
MA2100      10     0.50  1982-01-01  1982-11-01
MA2100      20     1.00  1982-01-01  1982-03-01
MA2110      10     1.00  1982-01-01  1983-02-01
MA2111      40     1.00  1982-01-01  1983-02-01
MA2111      50     1.00  1982-01-01  1982-06-01
MA2111      60     1.00  1982-06-01  1983-02-01
MA2111      60     1.00  1982-06-15  1983-02-01
MA2112      60     2.00  1982-01-01  1982-07-01
MA2112      70     1.00  1982-02-01  1982-10-01
MA2112      70     1.50  1982-02-15  1983-02-01
MA2112      70     1.00  1982-06-01  1983-02-01
MA2112      80     1.00  1982-10-01  1983-10-01
```

Figure 46. Verify Changed Dates in PROJ_ACT Table

If the changes are not correct, update again by issuing the following command:

```
rollback
```

If the changes are correct, you can type the COMMIT statement to commit the changes to the application server. Instead, to maintain the current sample data, type:

```
rollback
```

Updating Multiple Rows

Suppose you decide to give all employees an extra $25 bonus. Type the following:

```
update employee -
set bonus=bonus+25.00
```

The system returns a warning message indicating that more than one data row is affected. You receive this message because you did not include a WHERE clause in the UPDATE statement. Because you want to update all rows, you can ignore the message.

Select EMPNO, FIRSTNME, SALARY, BONUS, and COMM from the EMPLOYEE table to view the changes. The table now resembles Figure 47.

```
EMPNO   FIRSTNME          SALARY        BONUS         COMM
------  ------------  -----------  -----------  -----------
000010  CHRISTINE       52750.00      1025.00      4220.00
000110  VINCENZO        46500.00       925.00      3720.00
000120  SEAN            29250.00       625.00      2340.00
000020  MICHAEL         41250.00       825.00      3300.00
000030  SALLY           38250.00       825.00      3060.00
000130  DOLORES         23800.00       825.00      1904.00
000140  HEATHER         28420.00       825.00      2274.00
000060  IRVING          32250.00       625.00      2580.00
000150  BRUCE           25280.00       525.00      2022.00
000160  ELIZABETH       22250.00       425.00      1780.00
000170  MASATOSHI       24680.00       525.00      1974.00
000180  MARILYN         21340.00       525.00      1707.00
000190  JAMES           20450.00       425.00      1636.00
000200  DAVID           27740.00       625.00      2217.00
000210  WILLIAM         18270.00       425.00      1462.00
000220  JENNIFER        29840.00       625.00      2387.00
000070  EVA             36170.00       725.00      2893.00
000230  JAMES           22180.00       425.00      1774.00
000240  SALVATORE       28760.00       625.00      2301.00
000250  DANIEL          19180.00       425.00      1534.00
000260  SYBIL            1725.00       325.00      1380.00
000270  MARIA           27380.00       525.00      2190.00
```

Figure 47. Verify UPDATE of Multiple Rows

Now, type:

```
rollback
```

Updating Using Special Registers

You can assign any of the special registers to a column. For example, you can assign special registers to columns when the date or time value that you want is the current one (the date or time at which you are assigning the value to a column). The following example stores the current date in the designated row of the EMPLOYEE table:

```
update employee -
set hiredate = current date -
where empno = '000010'
```

You can also use CURRENT SERVER in an UPDATE or INSERT statement to put the name of the currently connected application server in a table. The following example shows how to use the INSERT statement to put the name of the application server to which your terminal is connected into column INRDB of table SAMP1:

```
insert into samp1 (id,indate,intime,inrdb) -
values (4,'06/24/1986','16:24', current server)
```

Updating Columns That Are Primary Keys, Foreign Keys, or Unique Constraints

The previous update examples contain columns that are not primary or foreign keys, and for which no particular update rules apply. For key columns in a parent table, a dependent table, or a column with the UNIQUE attribute, follow these rules:

- A new primary key value or a new unique constraint value can neither duplicate an existing primary key or unique constraint value in that table, nor can it be null.
- Primary key and unique constraint values *cannot* be changed to null.
- Primary key values that have matching dependent foreign key values *cannot* be changed.
- Any new foreign key value in a dependent table must match an existing primary key value in its parent or be null. A foreign key value can be updated to a null value only if its definition allows one or more of the columns making up the key to be set to null.

You may need to change the foreign key value for a row. For example, change the employee working on project IF2000, activity 100, which started on 1982-03-01, from employee number 140 to employee number 130 as follows:

```
update emp_act -
set empno = '000130' -
where empno = '000140' -
and projno = 'IF2000' -
and actno = 100 -
and emstdate = '1982-03-01'
```

Do a select on EMP_ACT for activities related to project number IF2000 to verify that the update worked. If the new employee number for the project activity was 40, this update would not occur. There is currently no employee number 40 in the EMPLOYEE table.

Primary key values are not likely to be changed often. Any change you do make is checked to ensure that it does not violate one of the above rules. For example, department B01 in the DEPARTMENT table is referenced by foreign keys in the EMPLOYEE table and in the PROJECT table. Try to change the department number for Planning from B01 to B11 by typing the following statement:

```
update department -
set deptno = 'B11' -
where deptno = 'B01'
```

Because there are matching dependent foreign key values, the system returns the following error message:

```
ARI0503E An SQL error has occurred.
         The DELETE or UPDATE operation is prevented by a rule of
         RESTRICT associated with the foreign key SQLDBA.R_DEPT2
ARI0505I SQLCODE = -532 SQLSTATE = 23504 ROWCOUNT = 0
ARI0504I SQLERRP: ARIXRID SQLERRD1: -160 SQLERRD2: 0
ARI7933I The ROLLBACK process is complete.
```

To change a primary key value that has matching foreign key values, you can:

1. Insert a new row for the new primary key value in the parent table.
2. Update the foreign key values that match the primary key value that you want to change. These foreign key values can be changed to the new primary key value, another existing primary key value, or to null (if allowed).
3. Delete the row containing the primary key value that you wanted to change.

Deleting Table Data

This section explains how to:

- Delete a row of information from a table
- Delete a row of information from a dependent table.

Deleting Rows

If you have not entered SET AUTOCOMMIT OFF, do so now. See “Using the AUTOCOMMIT OFF Setting” on page 71.

Delete information from a table by row. Individual columns cannot be deleted. For example, type:

```
delete from emp_act -
where projno = 'IF1000'
```

A message is displayed indicating that several rows are affected by the command.

Query the EMP_ACT table and compare the results with Figure 48 on page 80.

```
 EMPNO   PROJNO   ACTNO  EMPTIME  EMSTDATE    EMENDATE
 ------  ------  ------  -------  ----------  ----------
 000270  AD3113      60     0.25  1982-09-01  1982-10-15
 000270  AD3113      70     0.75  1982-09-01  1982-10-15
 000270  AD3113      70     1.00  1982-10-15  1983-02-01
 000270  AD3113      80     1.00  1982-01-01  1982-03-01
 000270  AD3113      80     0.50  1982-03-01  1982-04-01
 000030  IF2000      10     0.50  1982-01-01  1983-01-01
 000140  IF2000     100     1.00  1982-01-01  1982-03-01
 000130  IF2000     100     0.50  1982-03-01  1982-07-01
 000140  IF2000     110     0.50  1982-03-01  1982-07-01
 000140  IF2000     110     0.50  1982-10-01  1983-01-01
 000010  MA2100      10     0.50  1982-01-01  1982-11-01
 000110  MA2100      20     1.00  1982-01-01  1982-03-01
 000010  MA2110      10     1.00  1982-01-01  1983-02-01
 000200  MA2111      50     1.00  1982-01-01  1982-06-15
 000200  MA2111      60     1.00  1982-06-15  1983-02-01
 000220  MA2111      40     1.00  1982-01-01  1983-02-01
 000150  MA2112      60     1.00  1982-01-01  1982-07-15
 000150  MA2112     180     1.00  1982-07-15  1983-02-01
 000170  MA2112      60     1.00  1982-01-01  1983-06-01
 000170  MA2112      70     1.00  1982-06-01  1983-02-01
 000190  MA2112      70     1.00  1982-02-01  1983-10-01
 000190  MA2112      80     1.00  1982-10-01  1983-10-01
```

Figure 48. Verify That Rows Were Deleted

If a DELETE statement is typed without a WHERE clause, all rows of the table are deleted. The table still exists but no longer contains any rows. A message informing you of the absence of the WHERE clause is displayed so that you can catch potential deletions.

Deleting Data from a Table Containing Referential Constraints

The referential constraints in the sample tables, described in Appendix B, "Sample Tables" on page 269, provide examples of all three delete rules. These relationships and their associated delete rules are illustrated in Figure 44 on page 74. Because of a delete rule of RESTRICT, a row in the DEPARTMENT table cannot be deleted if its DEPARTMENT value is referenced in the PROJECT table. Deleting an employee who is also a manager of a department removes the employee (and employee number EMPNO) from the EMPLOYEE table, and sets that department's manager number to null in the DEPARTMENT table because of the delete rule of SET NULL. Deleting an employee from the EMPLOYEE table deletes rows for that employee in the EMP_ACT table because of the delete rule of CASCADE.

To see the effects of the delete rule of SET NULL, assume Irving Stern, employee number 60 and manager of department D11, has resigned. Delete this row in the EMPLOYEE table with the following command:

```
delete from employee -
where empno = '000060'
```

When the deletion is completed, the system displays a message informing you that two dependent rows are affected by the deletion (one in the parent table and the other in the dependent table). This message is displayed following deletions from a parent table in a referential constraint with a delete rule of SET NULL or of CASCADE. The message helps you to monitor the effects of parent row deletions.

Query the EMPLOYEE table and verify the results. Query the DEPARTMENT table as well. There should be a null value in the manager number column for department D11.

Inserting Table Data

To insert information in a table, you can enter either several rows at a time or one row at a time.

Inserting Multiple Rows

To use the ISQL command INPUT to insert multiple rows, type:

```
input activity
```

A display appears similar to Figure 49.

```
input activity
ARI7307A Enter data values, separated by commas,
         for the following columns.
         Character data must be enclosed in single quotes.
  COLUMN NAME                     DATA TYPE           NULL VALUES ALLOWED
    ACTNO                         WHOLE NUMBER             No
    ACTKWD                        CHARACTER (    6)        No
    ACTDESC                       CHARACTER (   20)        No
```

Figure 49. ISQL Display for Inserting Multiple Rows into a Table

Notice the message on the line below your statement in the output area. It explains how to enter a row of data. The data must be entered in the order listed (ACTNO, ACTKWD, and ACTDESC). This order was defined when the ACTIVITY table was created. You can specify a different order by using column names with the INPUT command like this:

```
input activity (actno,actdesc,actkwd)
```

Similarly, to insert data only in the ACTNO and ACTKWD columns, you can use:

```
input activity (actno,actkwd)
```

Returning to the message on your display, note that the type of data that belongs in each column is also described. Under the heading `NULL VALUES ALLOWED`, the word `NO` appears for all columns. This indicates that you must enter data in these columns. (A `YES` indicates you can omit data by typing *null* in the column.) This condition is determined by the owner of the table.

Now that you understand the message, type the first row of data:

```
1,'frst','First activity'
```

Your input line is moved to the output area as shown in Figure 50 on page 82.

```
input activity
ARI7307A Enter data values, separated by commas,
         for the following columns.
         Character data must be enclosed in single quotes.
  COLUMN NAME                     DATA TYPE           NULL VALUES ALLOWED
    ACTNO                         WHOLE NUMBER             No
    ACTKWD                        CHARACTER (    6)        No
    ACTDESC                       CHARACTER (   20)        No
1,'frst','First activity'
```

Figure 50. Display of Input Line Moved to Output Area

The database manager then waits for more input. Type the next row of data:

2,'scnd','Second activity'

Again, your input line moves to the output area and the database manager waits for you to type more input. This time, however, you have finished entering data and must inform the database manager. Type the following command:

end

A message appears prompting you to commit the new information. Because you should not alter the sample tables at this time, do not commit the new information. Type:

rollback

You have now finished adding rows. It is important to note that data added with an INPUT command is not committed to the table until the command is ended with the END command. It is not committed even then if AUTOCOMMIT is off. With AUTOCOMMIT on, you can commit data to the table before having entered all the new data. This is done by typing the following command in the input area instead of a row of data:

save

The SAVE command commits all data entered since the previous SAVE command or, if one had not been entered, since the start of the INPUT command. This is usually done periodically when you are entering large amounts of data.

In addition to committing data before ending an INPUT command, you can also delete all data entered since the last SAVE command, or since the start of the INPUT command if a SAVE command has not been entered. This is done by typing the following command in the input area instead of data:

backout

Use this if you discover typing errors in a row of data already moved to the output area.

Note: You can enter DBCS data into GRAPHIC and VARGRAPHIC columns. For example, you would type:

G'so...si'

to enter a DBCS string, where so is the shift-out character (X'0E'), si is the shift-in character (X'0F'), and ... is the DBCS string.

Note: You can use N' as a synonym for G'

Inserting a Single Row

You use the SQL statement INSERT to insert a single row. Continuing with the ACTIVITY table, type:

```
insert into activity -
values (3,'thrd','Third activity')
```

A message appears, indicating that processing was successful. Type:

```
rollback
```

As with the INPUT command, if you do not specify the columns in which data is to be added, addition is performed in the order defined when the table was created.

Inserting Data into Tables Containing Referential Constraints

Inserting Table Data

When inserting a single row or several rows into a table that is part of a referential structure, the following rules are enforced:

- Each row inserted into a parent table must have a non-null primary key value that does not duplicate an existing primary key value.
- Each row inserted into a dependent table must have a foreign key value that matches an existing primary key value in its parent table, or is null.

When you are inserting rows into both parent and dependent tables for new primary key values, you must insert the row containing the new primary key value into the parent table first. For example, a new employee who has been assigned to work on a project cannot be recorded in the EMP_ACT table until that employee's data has been inserted into the EMPLOYEE table.

As another example, assume a new project is approved, but its responsibility is not yet assigned to an employee. Insert a row for the new project with project number MA2114, project name Expert Systems, and department number D11. Leave the other fields null.

```
insert into project -
values ('MA2114','Expert Systems','D11',null,null,null,null,null)
```

Query the PROJECT table to verify that this row was entered.

Using Tables Containing Null Values

When querying columns that contain null values, use an IS NULL clause in the SELECT statement to retrieve only the null values in the query display.

The format for the IS NULL clause is:

```
►──column_name──IS──┬─────┬──NULL──────────────────────────────►
                    └─NOT─┘
```

As an example, type:

```
select * -
from department -
where mgrno = '000010' -
or mgrno is null
```

This produces Figure 51.

```
DEPTNO  DEPTNAME                              MGRNO   ADMRDEPT
------  ------------------------------------  ------  --------
A00     SPIFFY COMPUTER SERVICE DIV.          000010  A00
D01     DEVELOPMENT CENTER                    ?       A00
* End of Result *** 2 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 51. A Query Result with Null Values

The ? in the display represents a null value.

Use NOT with the NULL function to find the non-null values. To retrieve all information in the DEPARTMENT table that does not contain null values, type:

```
select * -
from department -
where mgrno is not null
```

Note: If any operand in an arithmetic operation is null, the result of the operation is null. For example, the result of *null + 6* is null. Now type:

```
rollback
```

To return to automatic statement committing by the database manager, type:

```
set autocommit on
```

Respond to the resulting message with:

```
rollback
```

EXERCISE 8 (Answers are in Appendix A, Answers to the Exercises, on page 263.)

Perform the following:

1. Start an LUW.
2. Insert the activity MARKETING, with the keyword MARKET and the number 190, into the ACTIVITY table.
3. Rollback your changes.
4. Retrieve all information for any item that has a null value in the major project column in PROJECT.
5. Update the PROJECT table to increase the mean project staff by 0.5, and make the project end date 7 days sooner.
6. Update the PROJECT table to show a project start date of March 1, 1982, for department E21 if the major project is OP2010.
7. Input into the DEPARTMENT table the following:
   - A department number of F01 with the name PERSONNEL, manager number 000110. This department reports to A00.
   - A department number of G01 with the name MARKETING AND SALES, manager number 000120. This department also reports to A00.
8. Delete the rows of the DEPARTMENT table for department numbers F01, F11, and F21.
9. Query the DEPARTMENT table, and then the PROJECT table, to verify your changes. Make any corrections necessary (or if you made serious errors, enter ROLLBACK and start over).
10. Insert Irving F. Stern's basic personnel information back into the EMPLOYEE table.

    This information is in Appendix B, "Sample Tables" on page 269 in the EMPLOYEE table, under employee number 000060.
11. Delete the new *Expert Systems* project (project number MA2114) from the PROJECT table.
12. To verify your changes, query the EMPLOYEE table and then query the PROJECT table.
13. Rollback your changes and return to committing commands automatically.

Chapter 6. Using ISQL Commands to Save Time When Executing Statements

To avoid retyping similar SQL statements or retyping an entire statement just to correct one error, you can use the ISQL commands in this chapter. It explains how to retrieve, modify, and delete SQL statements, how to withhold a statement from processing until you have checked it for errors, and how to reuse the same SQL statement for different tables, columns, and rows.

Reusing the Current SQL Statement

Before the system processes an SQL statement, it inserts it into a special storage area called the SQL *command buffer*. Once in the buffer, the statement becomes the current statement until it is pushed down in the queue by the next statement. ISQL commands are also inserted into the command buffer when you type them in, but not when you invoke them by using a PF key.

Using the command buffer facility, you can rerun a typed statement by using the ISQL START command (PF12).

Process the current SQL statement by performing the following:

1. Type the statement:

   ```
   select * -
   from department
   ```

2. When the result is displayed, type the END command.
3. Reenter the SQL statement by typing:

   ```
   start
   ```

The START command becomes even more useful when you have a typing error in an SQL statement. The error can be corrected (described in the next topic) and the START command used to reenter the corrected statement.

Retrieving and Correcting SQL Lines

You can use the RETRIEVE function to correct a statement already in the command line buffer. RETRIEVE moves the previously-typed line into the input area. The following example shows how to retrieve and correct a line in the output area. Type the following and press ENTER:

```
select actnum,actkwd,actdesc -
```

The line is moved to the output area.

Now type:

```
from activity
```

Press ENTER.

An error message is issued indicating that the column ACTNUM was not found. (ACTNO is the correct name.)

To retrieve the last input line entered, press the PF12 key to invoke the RETRIEVE function. The following line is now displayed in the input area with the cursor positioned at the end of the line:

```
from activity
```

Press the PF12 key again to retrieve the previous input line. The input area now contains:

```
select actnum,actkwd,actdesc -
```

with the cursor positioned at the end of the line. You can now correct the error as you would any other error in the input area by backspacing and typing the correct characters.

After you have corrected the line in the input area, press ENTER.

DB2 Server for VSE

The corrected line is now displayed in the output area, and `CONTINUE COMMAND` has appeared in the status area to confirm that you are to continue entering the statement.

DB2 Server for VM

The corrected line is now displayed in the output area, `VM READ` has appeared in the status area, and message `ARI7068I` has appeared in the output area to confirm that you are to continue typing the statement.

Continue the statement by using either of the following two methods:

- Type the FROM clause in the input area as follows:

  ```
  from activity
  ```

  Press ENTER.

- Press PF12 twice. The FROM clause is displayed in the input area with the cursor at the end of the line:

  ```
  from activity_
  ```

  Press ENTER.

The result is the same using either method: the SELECT statement is reissued.

Note: The command buffer holds a variable number of statements, depending on their length. If your statements are short, the buffer can hold more of them. As you press PF12 repeatedly, the system retrieves statements from the buffer starting with the most recent, and continues until it retrieves the oldest statement. Then it starts over and returns the most recent statement.

DB2 Server for VM

The RETRIEVE facility functions differently in wait mode compared to display mode. In display mode, the system can retrieve any information typed during the ISQL session that is still contained in the command buffer. In wait mode (`VM READ` shows in the status area of the display), the system can retrieve information typed in wait mode or from a time prior to starting ISQL.

Incorrect information typed from wait mode can be retrieved immediately, and corrected as illustrated in the previous example. Incorrect information typed from display mode forces display mode to end, and returns you to wait mode. Pressing PF12 does not retrieve the incorrect information in this case, because you typed it from display mode. The following examples illustrate the RETRIEVE-function differences between wait and display mode.

Ensure that you are in wait mode. If you do not see `VM READ` in the lower-right corner of your display, press PF3. Now, type the following (with *activity* spelled incorrectly):

```
select * from abtivity
```

An error message is issued indicating that ABTIVITY cannot be found. Press PF12. The incorrect `select * from abtivity` statement appears in your input area, where you can correct it.

Erroneous lines typed from display mode cannot be immediately retrieved with RETRIEVE key PF12. To illustrate, type the following correct statement from wait mode.

```
select * from activity
```

The ACTIVITY table is displayed, and the message disappears from the status area to indicate that you are in display mode.

Do not press PF3 to end the display. Instead, type the following incorrect statement from display mode:

```
select * from supply
```

The SUPPLY table is not found; `VM READ` reappears in the lower-right corner of your display to indicate that you are back in wait mode.

Now, press PF12. `select * from activity` appears in the input area. It is not the last statement you typed, but is the last one you typed from wait mode. The line in error that was entered from display mode cannot be retrieved from wait mode. It has been saved, however, and can be retrieved when you return to display mode. To illustrate this, enter the following correct query, which returns you to display mode:

```
select actno from activity
```

Now, from display mode, press PF12. The last line typed, `select actno from activity`, appears in the input area.

Press PF12 again. This time the line in error, `select * from supply`, does appear in the input area.

DB2 Server for VM

All lines typed during an ISQL session can be retrieved from within display mode (up to the maximum number of lines the buffer can hold), including lines entered from display mode and wait mode. Only those lines entered from wait mode can be retrieved from wait mode. Since ISQL commands other than display commands, SQL statements other than SELECT, and incorrect SELECT statements entered from display mode return you to wait mode, you cannot immediately retrieve this type of statement or command. Pressing PF3 after you have finished each query ensures that you enter each statement or command from wait mode and can immediately retrieve and change the information.

Altering and Reusing SQL Lines

The RETRIEVE facility is especially useful when you enter similar commands. To save time, you can retrieve an earlier command, alter it, and then enter it.

For example, enter the following statement:

```
select * -
from employee
```

Your next statement to be typed is `select * from department`. To save time, reuse the statement you just typed as follows.

Press PF12 twice to place `select * -` in the input area. Then, press ENTER.

Press PF12 twice. `from employee` is now contained in the input area. Backspace and type DEPARTMENT over EMPLOYEE. Then, press ENTER.

In this way, `select * from department` is typed in just a few keystrokes.

Remember, a *variable* number of your statements are retained, and you may or may not be able to retrieve a particular statement.

Changing the Current SQL Statement

These statements can be corrected, altered and reused, and portions of them can be deleted.

Correcting Typing Errors in the Statement

Having the facility to change the current SQL statement lets you correct one containing typing errors. For example, enter the following statement (with *activity* misspelled):

```
select actno -
from adtivity -
where actno > 100
```

This returns an error message stating that there is no table owned by you named ADTIVITY. Correct the error by entering the following ISQL command:

```
change /adtivity/activity/
```

The slashes in CHANGE commands separate the data to be changed (between the first two slashes) from the new data to replace it (between the last two slashes). Always include a blank before the first slash and always enter the final slash. ISQL changes the statement at the *first* occurrence of the data you place between the first two slashes, so ensure that the data you want changed is the first occurrence of that data in the statement to be changed.

Note: You do not have to use a slash to separate the data; any character except a blank can be used in its place. To change data that contains a slash, choose a character that does not occur in the data to be changed or in the new data.

Now enter the START command to process the changed SQL statement. If you have made no other mistakes, the query result for this SELECT statement is displayed on your display.

Altering and Reusing the Statement

You can use the CHANGE command to alter and then reuse the statement in the command buffer.

In "Altering and Reusing SQL Lines" on page 90, you used the RETRIEVE function to change the table name from EMPLOYEE to DEPARTMENT. Alternatively, you can query the EMPLOYEE table, end the query, and then type the following statement:

```
change /employee/department/
```

This command changes the statement in the buffer. To perform the new statement, which queries the DEPARTMENT table, type:

```
start
```

Notice that the RETRIEVE command uses *lines* of information from the command line buffer; the CHANGE command uses the *statement* from the command buffer.

Deleting Portions of the Statement

The CHANGE command can also be used to delete a portion of a current statement. For example, suppose you had selected the ACTNO, ACTKWD, and ACTDESC columns from the ACTIVITY table. After you end the query, you can delete the selection of the ACTKWD column table by typing:

```
change /,actkwd//
```

By providing no replacement string (nothing between the second and third slashes), the characters matching the string between the first two slashes are effectively deleted from the statement. You can see the result of your changed query by using the START command.

Ignoring an SQL Line

Another way to correct a typing mistake in a multiple-line statement is to use the IGNORE command.

To illustrate, type the following (*deptname* is misspelled):

```
select depname,mgrno -
```

Press ENTER. Now, you realize the valid column name is DEPTNAME. Type:

```
ignore
```

DB2 Server for VSE

You receive the message:

```
ARI7061I Previous input ignored.
```

in the output area, and the status area contains *ENTER A NEW COMMAND*.

DB2 Server for VM

You receive the message:

```
ARI7061I Previous input ignored.
```

in the output area, and the status area contains `VM READ`.

You can now retype the statement, or use PF12 to retrieve the line for correction.

Preventing the Immediate Processing of an SQL Statement

You can prevent an SQL statement from being processed immediately after being typed. This lets you check the statement for typing errors before it is processed using a START command. It also allows an SQL statement containing *placeholders* to be placed in the SQL command buffer, and values to be substituted for the placeholders when the statement is started using the START command. (The START command and placeholders are discussed in the next section.)

To illustrate, the following example shows how to prevent the statement *SELECT * FROM PROJECT* from being processed immediately. Type the following ISQL command:

```
hold select * from project
```

This statement remains in the buffer as the current statement until you enter another SQL statement (or another HOLD command).

The HOLD command can also be invoked by pressing PF9. If you press PF9 instead of ENTER after typing an SQL statement, it is placed in the command buffer and is not processed.

Your held statement can then be processed using the START command.

Note: HOLD cannot be used with ISQL commands.

Using Placeholders in SQL Statements

You can form SQL statements that contain placeholders. They reserve areas in the statements, which are filled in when the statement is performed. One reason for doing this, for example, is to avoid typing an UPDATE statement for each table update. Suppose you want to update the EMPLOYEE table to reflect a $100 bonus increase for particular employees.

1. To prevent your changes from being automatically committed, type:

```
set autocommit off
```

This starts a logical unit of work. Now type:

```
hold update employee -
set bonus = &1 -
where empno = '&2'
```

The use of HOLD at the beginning of this example places the UPDATE statement in the command buffer without processing it.

In the example, &1 and &2 are the placeholders. The number following the ampersand refers to the sequence in which the placeholders are replaced. The database manager performs the replacement as follows: the first item of information replaces &1, the second replaces &2, and so on.

2. Start the UPDATE statement and supply the replacement information:

   ```
   start (bonus+100.00 '000010')
   ```

   This command adds a $100 bonus to employee 000010, and produces a message indicating that one row (ROWCOUNT=1) was updated. The actual UPDATE statement processed is:

   ```
   update employee
   set bonus = bonus + 100.00
   where empno = '000010'
   ```

   The two items of information which replace the placeholders are called *parameters*.

3. Because START is an ISQL command, it does not replace the UPDATE statement in the command buffer. This lets you continue to use the UPDATE statement to update another row as follows:

   ```
   start (bonus+100.00 '000050')
   ```

4. This process can continue for as many updates as needed. Remember though, that the START command uses the statement currently contained in the command buffer. If you typed another SQL statement, it would replace the UPDATE statement in the buffer, and you would have to recall the UPDATE statement to continue. (Statement recalling is discussed in Chapter 8, “Storing SQL Statements” on page 119.)

5. Type the following command to return to normal command processing:

   ```
   set autocommit on
   ```

   Respond to the resulting message with the following to prevent the changes from being committed:

   ```
   rollback
   ```

The following are rules for the use of placeholders and parameters:

- A placeholder for a character data item must be enclosed in single quotation marks.
- A parameter for a character data item must also be enclosed in single quotation marks.
- When using parameters in a START command, enclose them in parentheses and separate each parameter with a blank.
- A parameter may be one word, several words, or an expression.

- If one parameter (replacing just one placeholder) consists of a list of words separated by commas, such as a list of column names in a *select_list*, the list of words must use commas (and not blanks) as separators.
- If a parameter contains a list of words separated by blanks, that list of words must be enclosed in single quotation marks to distinguish between the blanks within the one parameter and the blanks that *separate* parameters.

- In a stored SQL command, you can only use the ampersand (&) to create placeholders.

EXERCISE 9 (Answers are in Appendix A, Answers to the Exercises, on page 264.)

Perform the following:

1. Enter and hold an SQL statement that retrieves the entire DEPARTMENT table.
2. Change the command created in step 1 to select two columns, and use placeholders to define them.
3. Start the command, replacing the two placeholders with values that retrieve the DEPTNO and DEPTNAME columns. Check the results, and then end the display.
4. Change the current SQL statement to add a WHERE clause. Use a placeholder for the search condition.
5. Start the command, replacing the placeholders with values that select the department name and the manager number for departments that report to E01.

Chapter 7. Formatting Query Results

This chapter builds on the information in previous chapters. In Chapter 2, "Querying Tables" on page 21 and Chapter 3, "Manipulating Query Information" on page 41, you learned how to produce and manipulate query information. When the information appears on your display, you can change the way it looks. You can format the columns so that they appear with different headings, widths, or separators. You can format the whole query result so that it appears as a report, divided into groups, totaled, and titled. You can also use ISQL commands to set the query format characteristics for all queries in a session.

Formatting Columns

In Chapter 2, "Querying Tables" on page 21, you learned how to produce query information. In this chapter, you learn techniques to arrange the information in a report format. You can:

- Change the number of blanks that separate columns or even change the blanks to some other characters.
- Cease display of a column (and later include it, if desired).
- Change the name of a displayed column heading.
- Specify the number of decimal places to display for numeric columns.
- Control the display of leading zeros on numeric columns.
- Change the display width of a column.

Creating a Report from Query Results

To illustrate the formatting changes that you can make to columns, you first need a query result. Type the following statement:

```
select projno,actno,acstaff,acstaff + .25,acstdate -
from proj_act -
where projno = 'AD3100' -
or projno = 'AD3111' -
or projno = 'AD3112' -
order by projno,actno
```

The query result for this statement (shown in Figure 52 on page 96) is used over the next several topics. Do not end it until asked to do so.


```
PROJNO   ACTNO  ACSTAFF    EXPRESSION 1  ACSTDATE
------  ------  -------  --------------  ----------
AD3100      10     0.50            0.75  1982-01-01
AD3111      60     0.80            1.05  1982-01-01
AD3111      60     0.50            0.75  1982-03-15
AD3111      70     1.50            1.75  1982-02-15
AD3111      70     0.50            0.75  1982-03-15
AD3111      80     1.25            1.50  1982-04-15
AD3111      80     1.00            1.25  1982-09-15
AD3111     180     1.00            1.25  1982-10-15
AD3112      60     0.75            1.00  1982-01-01
AD3112      60     0.50            0.75  1982-02-01
AD3112      60     0.75            1.00  1982-12-01
AD3112      60     1.00            1.25  1983-01-01
AD3112      70     0.75            1.00  1982-01-01
AD3112      70     0.50            0.75  1982-02-01
AD3112      70     1.00            1.25  1982-03-15
AD3112      70     0.25            0.50  1982-08-15
AD3112      80     0.35            0.60  1982-08-15
AD3112      80     0.50            0.75  1982-10-15
AD3112     180     0.50            0.75  1982-08-15
* End of Result *** 19 Rows Displayed ***Cost Estimate is 1********************
```

Figure 52. A Query Result to Be Used to Illustrate Formatting Techniques

Field procedures can affect the order of the rows when you use the ORDER BY clause. For more information about field procedures, see the *DB2 Server for VSE & VM SQL Reference* manual.

Modifying the Separation between Columns

Using the query result described above, change the number of blanks used to separate the columns by typing the following display command:

```
format separator 4 blanks
```

FORMAT is the name of the command; SEPARATOR describes the kind of formatting to be done. After this command has been processed, four blanks are displayed between columns. Now separate the columns with a vertical bar and a couple of blanks by typing the following command:

```
format separator ' | '
```

This defines a separator that consists of a blank, followed by a vertical bar, followed by a blank. The quotation marks were used to include the blanks as part of the separator. Quotation marks are needed whenever the separator you are defining contains a blank. For example, the 3-character separator:

```
|||
```

does not require quotation marks, whereas this separator does:

```
'| |'
```

The result of typing the above FORMAT command is shown in Figure 53 on page 97.

```
 PROJNO │  ACTNO │ ACSTAFF │   EXPRESSION 1 │ ACSTDATE   │
 ------ │ ------ │ ------- │ -------------- │ ---------- │
 AD3100 │     10 │    0.50 │           0.75 │ 1982-01-01 │
 AD3111 │     60 │    0.80 │           1.05 │ 1982-01-01 │
 AD3111 │     60 │    0.50 │           0.75 │ 1982-03-15 │
 AD3111 │     70 │    1.50 │           1.75 │ 1982-02-15 │
 AD3111 │     70 │    0.50 │           0.75 │ 1982-03-15 │
 AD3111 │     80 │    1.25 │           1.50 │ 1982-04-15 │
 AD3111 │     80 │    1.00 │           1.25 │ 1982-09-15 │
 AD3111 │    180 │    1.00 │           1.25 │ 1982-10-15 │
 AD3112 │     60 │    0.75 │           1.00 │ 1982-01-01 │
 AD3112 │     60 │    0.50 │           0.75 │ 1982-02-01 │
 AD3112 │     60 │    0.75 │           1.00 │ 1982-12-01 │
 AD3112 │     60 │    1.00 │           1.25 │ 1983-01-01 │
 AD3112 │     70 │    0.75 │           1.00 │ 1982-01-01 │
 AD3112 │     70 │    0.50 │           0.75 │ 1982-02-01 │
 AD3112 │     70 │    1.00 │           1.25 │ 1982-03-15 │
 AD3112 │     70 │    0.25 │           0.50 │ 1982-08-15 │
 AD3112 │     80 │    0.35 │           0.60 │ 1982-08-15 │
 AD3112 │     80 │    0.50 │           0.75 │ 1982-10-15 │
 AD3112 │    180 │    0.50 │           0.75 │ 1982-08-15 │
 * End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 53. A Query Result with a Formatted Separator between Columns

Excluding Columns from the Display

There can be occasions when you obtain a query result and decide that it contains columns that you do not want in your report. You can end the result and retype the statement with the correct columns listed in the SELECT clause, but there is an alternative to retyping.

Suppose you want to exclude the ACSTAFF and ACSTDATE columns in the current display. Type:

```
format exclude (acstaff acstdate)
```

Issuing a FORMAT command with EXCLUDE instructs ISQL to exclude the columns specified from the current display. When specifying more than one column name, enclose them in parentheses and separate the names with a blank. The above FORMAT command displays Figure 54 on page 98.

```
PROJNO | ACTNO  |   EXPRESSION 1 |
------ | ------ | -------------- |
AD3100 |     10 |           0.75 |
AD3111 |     60 |           1.05 |
AD3111 |     60 |           0.75 |
AD3111 |     70 |           1.75 |
AD3111 |     70 |           0.75 |
AD3111 |     80 |           1.50 |
AD3111 |     80 |           1.25 |
AD3111 |    180 |           1.25 |
AD3112 |     60 |           1.00 |
AD3112 |     60 |           0.75 |
AD3112 |     60 |           1.00 |
AD3112 |     60 |           1.25 |
AD3112 |     70 |           1.00 |
AD3112 |     70 |           0.75 |
AD3112 |     70 |           1.25 |
AD3112 |     70 |           0.50 |
AD3112 |     80 |           0.60 |
AD3112 |     80 |           0.75 |
AD3112 |    180 |           0.75 |
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 54. A Query Result Formatted to Exclude Two Columns

You can use a number instead of a column name to identify the excluded column. The number refers to the column's position in the SELECT clause. For example, the ACSTAFF and ACSTDATE columns can be excluded by typing:

format exclude (3 5)

Sometimes it is easier to define columns you want *included*. For example, if you want to include only the third column of a query result that contained many columns, you can type:

format exclude all but (3)

The parentheses can be omitted when only one column is specified.

Including Columns in the Display

The effects of excluding a column from the display can be reversed. You can use the INCLUDE option to include the ACSTAFF and ACSTDATE columns in the current display. Do this by typing:

format include (acstaff acstdate)

Again, numbers can be used instead of column names. For example, to include only the first and third columns of a query result, you can type:

format include only (1 3)

The above FORMAT command displays Figure 55 on page 99.

```
PROJNO │ ACSTAFF │
------   -------
AD3100 │    0.50 │
AD3111 │    0.80 │
AD3111 │    0.50 │
AD3111 │    1.50 │
AD3111 │    0.50 │
AD3111 │    1.25 │
AD3111 │    1.00 │
AD3111 │    1.00 │
AD3112 │    0.75 │
AD3112 │    0.50 │
AD3112 │    0.75 │
AD3112 │    1.00 │
AD3112 │    0.75 │
AD3112 │    0.50 │
AD3112 │    1.00 │
AD3112 │    0.25 │
AD3112 │    0.35 │
AD3112 │    0.50 │
AD3112 │    0.50 │
* End of Result *** 19 Rows Displayed ***Cost Estimate is 1********************
```

Figure 55. A Query Result Formatted for Include-Only Columns

To display all the columns again, type:

format include

Changing a Displayed Column Heading

The query result you are using has EXPRESSION 1 as a column heading. To make the heading more meaningful, use the NAME keyword of the FORMAT command. Type the following command:

format column 'expression 1' name 'staff + .25'

The above FORMAT command displays Figure 56.

```
PROJNO │  ACTNO │ ACSTAFF │    STAFF + .25 │ ACSTDATE   │
------   ------   -------   --------------   ----------
AD3100 │     10 │    0.50 │           0.75 │ 1982-01-01 │
AD3111 │     60 │    0.80 │           1.05 │ 1982-01-01 │
AD3111 │     60 │    0.50 │           0.75 │ 1982-03-15 │
AD3111 │     70 │    1.50 │           1.75 │ 1982-02-15 │
AD3111 │     70 │    0.50 │           0.75 │ 1982-03-15 │
AD3111 │     80 │    1.25 │           1.50 │ 1982-04-15 │
AD3111 │     80 │    1.00 │           1.25 │ 1982-09-15 │
AD3111 │    180 │    1.00 │           1.25 │ 1982-10-15 │
AD3112 │     60 │    0.75 │           1.00 │ 1982-01-01 │
AD3112 │     60 │    0.50 │           0.75 │ 1982-02-01 │
AD3112 │     60 │    0.75 │           1.00 │ 1982-12-01 │
AD3112 │     60 │    1.00 │           1.25 │ 1983-01-01 │
AD3112 │     70 │    0.75 │           1.00 │ 1982-01-01 │
AD3112 │     70 │    0.50 │           0.75 │ 1982-02-01 │
AD3112 │     70 │    1.00 │           1.25 │ 1982-03-15 │
AD3112 │     70 │    0.25 │           0.50 │ 1982-08-15 │
AD3112 │     80 │    0.35 │           0.60 │ 1982-08-15 │
AD3112 │     80 │    0.50 │           0.75 │ 1982-10-15 │
AD3112 │    180 │    0.50 │           0.75 │ 1982-08-15 │
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 56. A Query Result Formatted to Change a Displayed Column Heading

Numbers can also be used to identify the column heading to change; for example:

```
format column 4 name 'staff + .25'
```

Remember, the 4 refers to the column's position in the SELECT clause, not the position of the column displayed.

Changing the Number of Decimal Places Displayed

The query result on your display shows two decimal places for both the ACSTAFF and ACSTAFF + .25 columns. You control the number of decimal places displayed using the DPLACES option of the FORMAT command. For example, to display only one decimal place for the STAFF + .25 column, type:

```
format column 'staff + .25' dplaces 1
```

This displays a result similar to Figure 57.

```
 PROJNO | ACTNO | ACSTAFF |    STAFF + .25 | ACSTDATE   |
 ------   ------   -------   --------------   ----------
 AD3100       10      0.50              0.7   1982-01-01
 AD3111       60      0.80              1.0   1982-01-01
 AD3111       60      0.50              0.7   1982-03-15
 AD3111       70      1.50              1.7   1982-02-15
 AD3111       70      0.50              0.7   1982-03-15
 AD3111       80      1.25              1.5   1982-04-15
 AD3111       80      1.00              1.2   1982-09-15
 AD3111      180      1.00              1.2   1982-10-15
 AD3112       60      0.75              1.0   1982-01-01
 AD3112       60      0.50              0.7   1982-02-01
 AD3112       60      0.75              1.0   1982-12-01
 AD3112       60      1.00              1.2   1983-01-01
 AD3112       70      0.75              1.0   1982-01-01
 AD3112       70      0.50              0.7   1982-02-01
 AD3112       70      1.00              1.2   1982-03-15
 AD3112       70      0.25              0.5   1982-08-15
 AD3112       80      0.35              0.6   1982-08-15
 AD3112       80      0.50              0.7   1982-10-15
 AD3112      180      0.50              0.7   1982-08-15
 * End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 57. A Query Result Formatted to Display One Decimal Place

Controlling the Display of Leading Zeros

You can show leading zeros for a numeric column. Do this for the ACTNO column by typing:

```
format column actno zeros on
```

The above FORMAT command displays Figure 58 on page 101.

```
PROJNO | ACTNO | ACSTAFF |   STAFF + .25 | ACSTDATE   |
------   ------   -------  --------------  ----------
AD3100 | 00010 |    0.50 |           0.7 | 1982-01-01 |
AD3111 | 00060 |    0.80 |           1.0 | 1982-01-01 |
AD3111 | 00060 |    0.50 |           0.7 | 1982-03-15 |
AD3111 | 00070 |    1.50 |           1.7 | 1982-02-15 |
AD3111 | 00070 |    0.50 |           0.7 | 1982-03-15 |
AD3111 | 00080 |    1.25 |           1.5 | 1982-04-15 |
AD3111 | 00080 |    1.00 |           1.2 | 1982-09-15 |
AD3111 | 00180 |    1.00 |           1.2 | 1982-10-15 |
AD3112 | 00060 |    0.75 |           1.0 | 1982-01-01 |
AD3112 | 00060 |    0.50 |           0.7 | 1982-02-01 |
AD3112 | 00060 |    0.75 |           1.0 | 1982-12-01 |
AD3112 | 00060 |    1.00 |           1.2 | 1983-01-01 |
AD3112 | 00070 |    0.75 |           1.0 | 1982-01-01 |
AD3112 | 00070 |    0.50 |           0.7 | 1982-02-01 |
AD3112 | 00070 |    1.00 |           1.2 | 1982-03-15 |
AD3112 | 00070 |    0.25 |           0.5 | 1982-08-15 |
AD3112 | 00080 |    0.35 |           0.6 | 1982-08-15 |
AD3112 | 00080 |    0.50 |           0.7 | 1982-10-15 |
AD3112 | 00180 |    0.50 |           0.7 | 1982-08-15 |
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 58. A Query Result Formatted to Display Leading Zeros

Stop the display of leading zeros in the ACTNO column by typing:

format column actno zeros off

Changing the Displayed Length Attribute of a Column

You may want to modify the displayed length attribute of a column to fit your report on the paper being used for printing. For example, to change the length attribute of the PROJNO column of the current query result, type:

format column projno width 8

The above FORMAT command displays Figure 59.

```
PROJNO   | ACTNO | ACSTAFF |   STAFF + .25 | ACSTDATE   |
--------   ------   -------  --------------  ----------
AD3100   |    10 |    0.50 |          0.75 | 1982-01-01 |
AD3111   |    60 |    0.80 |          1.05 | 1982-01-01 |
AD3111   |    60 |    0.50 |          0.75 | 1982-03-15 |
AD3111   |    70 |    1.50 |          1.75 | 1982-02-15 |
AD3111   |    70 |    0.50 |          0.75 | 1982-03-15 |
AD3111   |    80 |    1.25 |          1.50 | 1982-04-15 |
AD3111   |    80 |    1.00 |          1.25 | 1982-09-15 |
AD3111   |   180 |    1.00 |          1.25 | 1982-10-15 |
AD3112   |    60 |    0.75 |          1.00 | 1982-01-01 |
AD3112   |    60 |    0.50 |          0.75 | 1982-02-01 |
AD3112   |    60 |    0.75 |          1.00 | 1982-12-01 |
AD3112   |    60 |    1.00 |          1.25 | 1983-01-01 |
AD3112   |    70 |    0.75 |          1.00 | 1982-01-01 |
AD3112   |    70 |    0.50 |          0.75 | 1982-02-01 |
AD3112   |    70 |    1.00 |          1.25 | 1982-03-15 |
AD3112   |    70 |    0.25 |          0.50 | 1982-08-15 |
AD3112   |    80 |    0.35 |          0.60 | 1982-08-15 |
AD3112   |    80 |    0.50 |          0.75 | 1982-10-15 |
AD3112   |   180 |    0.50 |          0.75 | 1982-08-15 |
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 59. A Query Result Formatted to Display a Different Column Width

Columns defined as variable-character have an additional command to control the displayed length attribute. See the explanation of the VARCHAR keyword in the FORMAT command description in Chapter 13, "ISQL Commands" on page 203.

You have now finished formatting the report. To produce a copy, type a PRINT command before typing the END command.

You can specify more than one keyword in a single FORMAT command. For example, the following command combines several keywords:

```
format separator ' | ' column 'expression 1' name -
'staff + .25' dplaces 1 column actno zeros off
```

Because ISQL treats this information as a single command, considerable processing time is saved. Multiple-keyword entry is described in more detail under "Using More Than One Keyword in a FORMAT Command" on page 108.

EXERCISE 10 (Answers are in Appendix A, Answers to the Exercises, on page 264.)

Perform the following:

1. Retrieve the employee number, project number, and proportion of employee time from the EMP_ACT table for project numbers IF1000 and IF2000. Order the results primarily by project number and secondarily by employee number.
2. Separate all columns with two blanks, an asterisk, and two more blanks.
3. Display all columns except the EMPNO column.
4. Change the column heading for the EMPTIME column to PROPTN and its displayed length attribute to 8 characters.

Formatting Reports

Earlier in this chapter, you learned formatting techniques to create a report. In this section, you learn to prepare totals, create an outline format, and specify report titles.

To illustrate these functions, type the following query statement and format commands:

```
select projno,actno,acstaff,acstaff + .25,acstdate -
from proj_act -
where projno = 'AD3100' -
or projno = 'AD3111' -
or projno = 'AD3112' -
order by projno,actno

format separator ' | ' column 'expression 1' name -
'staff + .25' -
column projno width 8
```

This produces the following result in Figure 60 on page 103.

```
PROJNO     ACTNO   ACSTAFF      STAFF + .25   ACSTDATE
--------   ------  -------   --------------   ----------
AD3100         10     0.50             0.75   1982-01-01
AD3111         60     0.80             1.05   1982-01-01
AD3111         60     0.50             0.75   1982-03-15
AD3111         70     1.50             1.75   1982-02-15
AD3111         70     0.50             0.75   1982-03-15
AD3111         80     1.25             1.50   1982-04-15
AD3111         80     1.00             1.25   1982-09-15
AD3111        180     1.00             1.25   1982-10-15
AD3112         60     0.75             1.00   1982-01-01
AD3112         60     0.50             0.75   1982-02-01
AD3112         60     0.75             1.00   1982-12-01
AD3112         60     1.00             1.25   1983-01-01
AD3112         70     0.75             1.00   1982-01-01
AD3112         70     0.50             0.75   1982-02-01
AD3112         70     1.00             1.25   1982-03-15
AD3112         70     0.25             0.50   1982-08-15
AD3112         80     0.35             0.60   1982-08-15
AD3112         80     0.50             0.75   1982-10-15
AD3112        180     0.50             0.75   1982-08-15
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 60. A Formatted Query Result to Be Used to Illustrate Report Format

Obtaining an Outline Report Format

An outline report format suppresses the display of duplicate values in a particular column. To provide this feature for the PROJNO column, type the following command:

```
format group (projno)
```

This produces Figure 61.

```
PROJNO     ACTNO   ACSTAFF      STAFF + .25   ACSTDATE
--------   ------  -------   --------------   ----------
AD3100         10     0.50             0.75   1982-01-01

AD3111         60     0.80             1.05   1982-01-01
               60     0.50             0.75   1982-03-15
               70     1.50             1.75   1982-02-15
               70     0.50             0.75   1982-03-15
               80     1.25             1.50   1982-04-15
               80     1.00             1.25   1982-09-15
              180     1.00             1.25   1982-10-15

AD3112         60     0.75             1.00   1982-01-01
               60     0.50             0.75   1982-02-01
               60     0.75             1.00   1982-12-01
               60     1.00             1.25   1983-01-01
               70     0.75             1.00   1982-01-01
               70     0.50             0.75   1982-02-01
               70     1.00             1.25   1982-03-15
               70     0.25             0.50   1982-08-15
               80     0.35             0.60   1982-08-15
               80     0.50             0.75   1982-10-15
              180     0.50             0.75   1982-08-15
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 61. A Query Result Displayed in Outline Report Format

Outlining is appropriate only on a column that has been ordered into groups of similar values (through an ORDER BY clause in the SELECT statement). Although outlining can be performed on an unordered column, the frequent changes in the values that are likely to occur in that column cause such outlining to be of little value.

Outlining is normally performed when you specify GROUP on a FORMAT command. For a description of how this process is controlled, see the FORMAT command section in Chapter 13, “ISQL Commands” on page 203.

Note: If you have VARCHAR or VARGRAPHIC columns with values that differ only by trailing blanks, the FORMAT GROUP command treats them as duplicates. Therefore, if you have 'AD3100' and 'AD3100 ' in a VARCHAR column, a FORMAT GROUP command eliminates one of them.

Obtaining Totals for Reports

To produce a total for the STAFF + .25 column, type:

format total ('staff + .25')

Note: Although included here, the parentheses are necessary only when specifying multiple columns to be totaled.

This FORMAT command displays a result similar to Figure 62.

```
PROJNO   | ACTNO | ACSTAFF |    STAFF + .25 | ACSTDATE   |
-------- | ------ | ------- | -------------- | ---------- |
AD3100   |     10 |    0.50 |           0.75 | 1982-01-01 |

AD3111   |     60 |    0.80 |           1.05 | 1982-01-01 |
         |     60 |    0.50 |           0.75 | 1982-03-15 |
         |     70 |    1.50 |           1.75 | 1982-02-15 |
         |     70 |    0.50 |           0.75 | 1982-03-15 |
         |     80 |    1.25 |           1.50 | 1982-04-15 |
         |     80 |    1.00 |           1.25 | 1982-09-15 |
         |    180 |    1.00 |           1.25 | 1982-10-15 |

AD3112   |     60 |    0.75 |           1.00 | 1982-01-01 |
         |     60 |    0.50 |           0.75 | 1982-02-01 |
         |     60 |    0.75 |           1.00 | 1982-12-01 |
         |     60 |    1.00 |           1.25 | 1983-01-01 |
         |     70 |    0.75 |           1.00 | 1982-01-01 |
         |     70 |    0.50 |           0.75 | 1982-02-01 |
         |     70 |    1.00 |           1.25 | 1982-03-15 |
         |     70 |    0.25 |           0.50 | 1982-08-15 |
         |     80 |    0.35 |           0.60 | 1982-08-15 |
         |     80 |    0.50 |           0.75 | 1982-10-15 |
         |    180 |    0.50 |           0.75 | 1982-08-15 |
         |        |         | ============== |            |
         |        |         |          18.65 |            |
* End of Result ************* 19 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 62. A Query Result Formatted to Produce Totals

You can also create subtotals for this column by typing the following command:

format subtotal ('staff + .25')

This displays Figure 63 on page 105.

```
PROJNO   | ACTNO | ACSTAFF |  STAFF + .25 | ACSTDATE   |
-------- | ----- | ------- | ------------ | ---------- |
AD3100   |    10 |    0.50 |         0.75 | 1982-01-01 |
         |       |         | ------------ |            |
******** |       |         |         0.75 |            |

AD3111   |    60 |    0.80 |         1.05 | 1982-01-01 |
         |    60 |    0.50 |         0.75 | 1982-03-15 |
         |    70 |    1.50 |         1.75 | 1982-02-15 |
         |    70 |    0.50 |         0.75 | 1982-03-15 |
         |    80 |    1.25 |         1.50 | 1982-04-15 |
         |    80 |    1.00 |         1.25 | 1982-09-15 |
         |   180 |    1.00 |         1.25 | 1982-10-15 |
         |       |         | ------------ |            |
******** |       |         |         8.30 |            |

AD3112   |    60 |    0.75 |         1.00 | 1982-01-01 |
         |    60 |    0.50 |         0.75 | 1982-02-01 |
         |    60 |    0.75 |         1.00 | 1982-12-01 |
         |    60 |    1.00 |         1.25 | 1983-01-01 |
         |    70 |    0.75 |         1.00 | 1982-01-01 |
         |    70 |    0.50 |         0.75 | 1982-02-01 |
         |    70 |    1.00 |         1.25 | 1982-03-15 |
         |    70 |    0.25 |         0.50 | 1982-08-15 |
```

Figure 63. A Query Result Formatted to Produce Subtotals

Notice that subtotals are created in the `STAFF + .25` column for each change in the value in the `PROJNO` column.

The conditions on which subtotals are created are defined, like outlining, by the GROUP option of a FORMAT command. Subtotals are created whenever the value changes in the column (or columns) specified with FORMAT GROUP. Columns identified with FORMAT GROUP should have been specified in the ORDER BY clause of the SELECT statement that produced the query results, and should appear in the same sequence as they appeared in the ORDER BY clause. Otherwise, the resulting subtotals can be meaningless.

You can erase these totals and subtotals. For example, to erase the totals in the above report, you would type:

```
format total erase
```

This FORMAT command would display a result similar to Figure 64 on page 106.

```
PROJNO   │ ACTNO │ ACSTAFF │   STAFF + .25 │ ACSTDATE   │
-------- │ ----- │ ------- │ ------------- │ ---------- │
         │    60 │    0.50 │          0.75 │ 1982-03-15 │
         │    70 │    1.50 │          1.75 │ 1982-02-15 │
         │    70 │    0.50 │          0.75 │ 1982-03-15 │
         │    80 │    1.25 │          1.50 │ 1982-04-15 │
         │    80 │    1.00 │          1.25 │ 1982-09-15 │
         │   180 │    1.00 │          1.25 │ 1982-10-15 │
         │       │         │ ------------- │            │
******** │       │         │          8.30 │            │

AD3112   │    60 │    0.75 │          1.00 │ 1982-01-01 │
         │    60 │    0.50 │          0.75 │ 1982-02-01 │
         │    60 │    0.75 │          1.00 │ 1982-12-01 │
         │    60 │    1.00 │          1.25 │ 1983-01-01 │
         │    70 │    0.75 │          1.00 │ 1982-01-01 │
         │    70 │    0.50 │          0.75 │ 1982-02-01 │
         │    70 │    1.00 │          1.25 │ 1982-03-15 │
         │    70 │    0.25 │          0.50 │ 1982-08-15 │
         │    80 │    0.35 │          0.60 │ 1982-08-15 │
         │    80 │    0.50 │          0.75 │ 1982-10-15 │
         │   180 │    0.50 │          0.75 │ 1982-08-15 │
         │       │         │ ------------- │            │
******** │       │         │          9.60 │            │
```

Figure 64. A Query Result Formatted with the Totals Erased

Subtotals can be erased and included in the same manner by substituting SUBTOTAL for TOTAL in the above example. Erasing subtotals also erases totals for the specified columns.

There are some variations in the use of GROUP, SUBTOTAL, and TOTAL with FORMAT commands. See the FORMAT command section in Chapter 13, “ISQL Commands” on page 203 for details.

Creating Titles for Printed Reports

Specify a top title for the current report by typing:

```
format ttitle 'summary of employee time'
```

The quotation marks are needed because the title contains blanks. The command displays the current top title, and prompts you to return to the query result by issuing the following message in the status area:

DB2 Server for VSE

```
Press clear key to continue
```

DB2 Server for VSE

```
MORE ...
```

To return to the query result, press CLEAR.

The top title can be erased and replaced with the first 100 characters of the associated SELECT statement by typing:

```
format ttitle erase
```

A bottom title can also be specified. Use the following command to create a bottom title for this report:

```
format btitle 'company confidential'
```

The bottom title can also be erased by typing:

```
format btitle erase
```

Top and bottom titles are centered in the top and bottom margins of the printed report. Although the titles cannot be seen until your report is printed, you can view them by typing FORMAT TTITLE (or FORMAT BTITLE) without specifying a title. For example, type:

```
format ttitle
```

Pressing CLEAR in response to this message returns you to the query result.

Now print a copy of this report by typing:

```
print
```

Your printed report is similar to Figure 65 on page 108.

```
.    08/10/89           SUMMARY OF EMPLOYEE TIME                    .
.                                                                   .
.                                                                   .
.    PROJNO   | ACTNO | ACSTAFF |    STAFF + .25 | ACSTDATE   |       .
.    _______  | _____ | _______ |  _____________ | __________ |       .
.    AD3100   |    10 |    0.50 |           0.75 | 1982-01-01 |       .
.             |       |         | ______________ |            |       .
.    ******** |       |         |           0.75 |            |       .
.                                                                   .
.    AD3111   |    60 |    0.80 |           1.05 | 1982-01-01 |       .
.             |    60 |    0.50 |           0.75 | 1982-03-15 |       .
.             |    70 |    1.50 |           1.75 | 1982-02-15 |       .
.             |    70 |    0.50 |           0.75 | 1982-03-15 |       .
.             |    80 |    1.25 |           1.50 | 1982-04-15 |       .
.             |    80 |    1.00 |           1.25 | 1982-09-15 |       .
.             |   180 |    1.00 |           1.25 | 1982-10-15 |       .
.             |       |         | ______________ |            |       .
.    ******** |       |         |           8.30 |            |       .
.                                                                   .
.    AD3112   |    60 |    0.75 |           1.00 | 1982-01-01 |       .
.             |    60 |    0.50 |           0.75 | 1982-02-01 |       .
.             |    60 |    0.75 |           1.00 | 1982-12-01 |       .
.             |    60 |    1.00 |           1.25 | 1983-01-01 |       .
.             |    70 |    0.75 |           1.00 | 1982-01-01 |       .
.             |    70 |    0.50 |           0.75 | 1982-02-01 |       .
.             |    70 |    1.00 |           1.25 | 1982-03-15 |       .
.             |    70 |    0.25 |           0.50 | 1982-08-15 |       .
.             |    80 |    0.35 |           0.60 | 1982-08-15 |       .
.             |    80 |    0.50 |           0.75 | 1982-10-15 |       .
.             |   180 |    0.50 |           0.75 | 1982-08-15 |       .
.             |       |         | ______________ |            |       .
.    ******** |       |         |           9.60 |            |       .
.             |       |         | ============== |            |       .
.             |       |         |          18.65 |            |       .
.                                                                   .
.                                                                   .
.                         COMPANY CONFIDENTIAL                      .
```

Figure 65. Example of a Printed Report

Using More Than One Keyword in a FORMAT Command

The following example provides additional practice in using multiple-keyword FORMAT commands.

Assume you want to create a report from a query result that is to include the following modifications:

- Change the blanks that separate columns to another character (SEPARATOR keyword).
- Change the name of a column heading (NAME keyword).
- Leave out a column (EXCLUDE keyword).
- Specify the number of decimal places to be displayed for decimal columns (DPLACES keyword).
- Create subtotals for specific columns (SUBTOTAL keyword and GROUP keyword).
- Create totals for specific columns (TOTAL keyword and GROUP keyword).

First, type the following statement:

```
select actno,acstaff,acstdate -
from proj_act -
where actno between 0 and 20 -
order by actno
```

This produces the display in Figure 66.

```
 ACTNO  ACSTAFF  ACSTDATE
------  -------  ----------
    10     0.50  1982-01-01
    10     1.00  1982-01-01
    10     0.25  1982-01-01
    10     1.00  1982-01-01
    10     0.50  1982-01-01
    10     0.50  1982-01-01
    10     0.50  1982-06-01
    10     0.50  1982-01-01
    10     1.00  1982-01-01
    10     1.00  1982-01-01
    20     1.00  1982-01-01
* End of Result *** 11 Rows Displayed ***Cost Estimate is 1********************
```

Figure 66. A Query Result to Be Used for a Multiple-Keyword Format Command

Using the above query result, type the following FORMAT command:

```
format column 2 name 'mean empl' dplaces 1 -
separator ' | ' exclude (acstdate) -
group actno -
subtotal ('mean empl')
```

The command produces Figure 67.

```
 ACTNO | MEAN EMPL |
------ | --------- |
    10 |       0.5 |
       |       1.0 |
       |       0.2 |
       |       1.0 |
       |       0.5 |
       |       0.5 |
       |       0.5 |
       |       0.5 |
       |       1.0 |
       |       1.0 |
       | --------- |
****** |       6.7 |

    20 |       1.0 |
       | --------- |
****** |       1.0 |
       | ========= |
       |       7.7 |

* End of Result ************* 11 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 67. A Query Result Formatted by a Multiple-Keyword Format Command

Use the END command to end this query and clear the display.

Displaying Null Values and Arithmetic Errors

Null values (indicated by a question mark) and arithmetic errors (indicated by number signs separated by blanks) are displayed in a formatted table. They are treated as zeros in any total or subtotal calculations of the columns they appear in.

In the sample report in Figure 68, the mean employee staff is unknown for one project activity, and contains an arithmetic error for another.

```
.          SUMMARY OF EMPLOYEE DISTRIBUTION FOR ACTIVITIES 10 AND 20    .
.                                                                       .
.                                                                       .
.   02/09/89                                               PAGE  1      .
.                                                                       .
.                                                                       .
.      ACTNO | MEAN EMPL |                                              .
.     ______ | _________ |                                              .
.         10 |       0.5 |                                              .
.            | # # # # # |                                              .
.            |       0.2 |                                              .
.            |       1.0 |                                              .
.            |       0.5 |                                              .
.            |       0.5 |                                              .
.            |       0.5 |                                              .
.            |         ? |                                              .
.            |       1.0 |                                              .
.            | _________ |                                              .
.     ****** |       6.7 |                                              .
.                                                                       .
.         20 |       1.0 |                                              .
.            | _________ |                                              .
.     ****** |       1.0 |                                              .
.            | ========= |                                              .
.            |       7.7 |                                              .
.                                                                       .
.                                                                       .
.                                                                       .
.                                                                       .
.                            COMPANY CONFIDENTIAL                       .
```

Figure 68. Sample Report Displaying Null Values and Arithmetic Errors

Controlling Null-Field Displays

When formatting reports, you may want something other than a question mark to be used to represent null values. To illustrate, type the following statement:

```
select *  -
from department -
order by mgrno
```

This displays Figure 69 on page 111.

```
DEPTNO  DEPTNAME             MGRNO   ADMRDEPT
------  --------------------  ------  --------
A00     SPIFFY COMPUTER SERV< 000010  A00
B01     PLANNING              000020  A00
C01     INFORMATION CENTER    000030  A00
E01     SUPPORT SERVICES      000050  A00
D11     MANUFACTURING SYSTEM< 000060  D01
D21     ADMINISTRATION SYSTE< 000070  D01
E11     OPERATIONS            000090  E01
E21     SOFTWARE SUPPORT      000100  E01
D01     DEVELOPMENT CENTER    ?       A00
* End of Result *** 9 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 69. A Query Result Displaying a Null Value

To format a report from this query result that replaces the question mark with **NULL**, type:

format null *null*

This FORMAT command should display a result similar to Figure 70.

```
DEPTNO  DEPTNAME             MGRNO   ADMRDEPT
------  --------------------  ------  --------
A00     SPIFFY COMPUTER SERV< 000010  A00
B01     PLANNING              000020  A00
C01     INFORMATION CENTER    000030  A00
E01     SUPPORT SERVICES      000050  A00
D11     MANUFACTURING SYSTEM< 000060  D01
D21     ADMINISTRATION SYSTE< 000070  D01
E11     OPERATIONS            000090  E01
E21     SOFTWARE SUPPORT      000100  E01
D01     DEVELOPMENT CENTER    *NULL*  A00
* End of Result *** 9 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 70. A Query Result Displaying a Formatted Null Field

The maximum number of characters that can be used as a null field indicator is 20.

Use the END command to end this query and clear the display.

Controlling Query Format Characteristics

You will probably develop a standard style for formatting query results that will be consistent across queries. Given this standardization, you can specify some formatting at the beginning of a display terminal session. The formatting remains effective for the session, unless you change or override it.

The formatting that you can set in this fashion is listed below:

- Punctuation displayed for numeric columns.
- Separation characters displayed between columns.
- Characters displayed for null fields.
- The displayed length attribute of variable character fields. This topic is explained in the description of the VARCHAR item in the FORMAT command section of Chapter 13, “ISQL Commands” on page 203.

The number of copies and page size can also be set for the duration of a display terminal session.

Note: Formatting information can also be set up automatically every time you begin a DB2 Server for VSE & VM display-terminal session, by defining the information in a routine. For an explanation of the routines, refer to “Profile Routines” on page 125.

Setting the Format Characteristics by Using the SET Command

ISQL gives you some control over what you see on your display. You can specify:

- Punctuation displayed for numeric fields
- Separation characters displayed between columns
- Characters displayed for null fields
- Page size of printed reports
- Language of messages and HELP text.

Furthermore, you can specify all of these features in one command and get a list of the current settings.

Punctuation Displayed for Numeric Fields

Punctuation for numeric fields refers to the use of periods, commas, and blanks for the decimal and thousands separators. Valid combinations of the decimal and thousands separators are:

Thousands Separator	Decimal Separator	Example
(nothing)	.	1234.56
,	.	1,234.56
.	,	1.234,56
(a blank)	,	1 234,56

You can set any of these combinations for the duration of a session. For example, set the thousands separator to a comma and the decimal separator to a period for the duration of the current session by typing:

```
set decimal /,/./
```

The character between the first two slashes represents the thousands separator; the character between the last two, the decimal separator. The slashes distinguish the thousands separator from the decimal separator in the command.

Now, observe how a number is displayed using these separators. Type the following statement:

```
select 1000 * acstaff -
from proj_act -
where projno = 'ma2112' and actno = 60 -
and acstdate = '1982-01-01'
```

This SELECT statement displays a result similar to Figure 71 on page 113.

```
  EXPRESSION 1
--------------
      2,000.00
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 71. A Query Result with a Formatted Numeric Field

This punctuation remains in effect for all numeric columns until the end of this session. Your next session begins with the normal default (no thousands separator and a period for the decimal separator).

The valid punctuation combinations are set like this:

```
set decimal /,/./
set decimal /./,/
set decimal / /,/
set decimal //./   (nothing for thousands separator)
```

Separation Characters Displayed between Columns

Using the ISQL SET command, you can set the number of blanks or the kinds of characters to be displayed between columns for the duration of a session. The syntax for this command is shown in the following diagram:

```
                         ┌─2───────┐
►►──SET──SEParator──┬─┼─────────┼─BLANKs─┬──────────────────────────►◄
                    │ └─integer─┘        │
                    └─string─────────────┘
```

This command can set the number of blanks displayed between the columns when an integer is used. For example, the following command causes five blanks to be displayed between columns:

```
set separator 5 blanks
```

You can set a character string to be displayed between columns. To display an asterisk between columns, type the following command:

```
set separator *
```

Characters Displayed for Null Fields

You can set the characters displayed for null fields for the duration of a session by typing a SET NULL command as illustrated in the following diagram:

```
              ┌─?──────┐
►►──SET──NULL─┼────────┼─────────────────────────────────────►◄
              └─string─┘
```

Replace `string` with the actual characters you want displayed. The maximum string length is 20 characters.

Number of Copies of Printed Reports (DB2 Server for VSE)

In addition to specifying the number of copies desired for printed reports on the PRINT command, you can specify the number of copies for *all* print requests you make during the current session. This command has the following syntax:

```
              ┌─1─┐
►►──SET──COPIES─┤   ├──────────────────────────────────────────────►◄
              └─n─┘
```

Replace `n` with the number of copies required. The maximum number that can be specified is 99.

If you specify the number of copies on the PRINT command after also having specified it using a SET command, the PRINT command quantity is used for that print operation. All following PRINT commands use the quantity specified by the SET command unless they too include the `COPIES` keyword.

Page Size of Printed Reports

Defining the page size lets you place printed query results on various paper sizes. Page size is defined in terms of the number of characters that are to be printed on a line and the number of lines that are to be printed on a page.

Before defining the page size, consider the output paper size to be used and the printer characteristics (characters per inch on a line and number of lines per vertical inch) to ensure that your definition fits on the paper. For example, suppose the printer class you are going to use is set up to use 8-1/2 inch wide by 11 inch long paper, and the printer prints 10 characters to the inch horizontally and prints lines at 6 to the inch vertically. This would allow each line to contain 85 characters (8.5 x 10) and 66 lines to be on a page. The maximum page size would therefore be 85 characters wide and 66 lines in length. The maximum number of lines that can actually be printed on a page is 8 less than the length (8 lines are reserved for top and bottom titles and column headings).

Once set, the specified page size remains in effect for the duration of the display terminal session or until it is changed. The SET PAGESIZE command has the following format:

```
                            ┌─132─────┐        ┌─66──────┐
►►──SET──PAGEsize──WIDth──┤         ├──LENgth──┤         ├──────►◄
                            └─integer─┘        └─integer─┘
```

The specified width must be from 19 to 204 and the length must be from 9 to 32767.

When setting the page size, it is not necessary to specify both width and length. In addition, length and width can be specified in either order. The default value for page size is a width of 132 and a length of 66.

Language of Messages and HELP Text

Messages and HELP text can be displayed in any of several national languages. If additional languages were installed on your system, you can use the SET command to display messages and HELP text in another language. Operator messages are displayed in the national language of the application server. The SET LANGUAGE command takes the following form:

```
►►──SET LANGuage──┬─language_name─┬──────────────────────────────────────►◄
                  └─langid────────┘
```

Messages can be displayed in one of the languages listed in the table in Figure 72.

Figure 72. Alternative Languages for Messages and HELP Text

Language	**Language ID**
American English (mixed case)	AMENG
American English (uppercase)	UCENG
French	FRANC
German	GER
Simplified Chinese	HANZI
Japanese	KANJI

Either the name of the language or the language ID can be used as the language identifier in the SET LANGUAGE command. For example, you could have messages displayed in French by typing:

```
set language franc
```

If your system uses *french* as the language name you could also type:

```
set language french
```

Languages for which there is no language ID can be set using the name of the language. If your system does not support the language you request, an error message is displayed in the default language.

To find out what languages are available on your system (and what language names or language identifiers you can use to select a language) type:

```
select *  from sqldba.syslanguage
```

A table is displayed listing the names, language IDs, and a brief description of each language available to you.

Multiple Format Characteristics

You can use the SET command with more than one keyword. In this way, you can specify or change multiple characteristics to be effective for the duration of your display session. For example, type the following command to combine several features that you set earlier using separate commands:

```
set autocommit on null *null* separator 2 blanks
```

This command sets AUTOCOMMIT to on, display **NULL** for each null field, and produces a display with columns separated by 2 blanks.

The range of characteristics you can set with the SET command is included in Chapter 13, “ISQL Commands” on page 203.

List of Current Settings

The current settings of all the format characteristics described above can be listed on your display. List them by typing:

```
list set *
```

The asterisk means that you want to list all settings. This displays a series of messages that describe the settings of each operational characteristic.

You can request a specific characteristic by specifying the name of the characteristic instead of the asterisk on the SET command. For example, type the following command to list the current setting for the column separator:

```
list set separator
```

Printing Reports on a Workstation Printer (DB2 Server for VSE)

Your printed output is automatically sent to the system printer designated by your site. You can change or redirect your printed output to a CICS terminal printer or POWER remote workstation.

To route your printed output to a CICS terminal printer, you would specify:

```
print termid  termid
```

or

```
set printroute  termid
```

where *termid* is replaced with the CICS terminal printer identifier.

To route printed output to a POWER remote workstation, specify:

```
print destid  wkstat
```

or

```
set printroute  wkstat
```

where *wkstat* is replaced with the identifier for the required POWER remote workstation.

The system printer is specified as:

```
print system
```

or

```
set printroute system
```

EXERCISE 11 (Answers are in Appendix A, Answers to the Exercises, on page 265.)

Perform the following:

1. For DB2 Server for VSE, set the number of copies requested for printed reports to two.
2. List the current settings for format characteristics.
3. Retrieve all of the information from PROJ_ACT where the project number is AD3112; order the result by activity end date.
4. Create an outline format for the activity end date column and exclude the activity start date column.
5. Create a top title called *PERSONNEL PROGRAMMING DEADLINES*.
6. For DB2 Server for VSE, request printed copies of this report.
7. For DB2 Server for VM, request two printed copies of this report.

Chapter 8. Storing SQL Statements

Storing frequently-used SQL statements saves you from retyping them. On longer statements, such as that used in the previous chapter to format a report, typing errors can be avoided by storing the statement and reusing it. A stored SQL statement may also contain placeholders.

Storing the Current Statement

Type the following:

```
select deptno,mgrno -
from department -
order by mgrno
```

Now, type the following format commands:

```
format separator ' | ' -
column mgrno width 8
```

End the query.

Store the current SQL statement by typing the following:

```
store dept
```

If you know that the stored command DEPT already exists, and you wish to replace it, include the keyword REPLACE in your STORE command, as follows:

```
store dept replace
```

When the STORE command is processed, a message informs you that the SQL statement is stored. Use storage names that are 8 characters or less in length. Do not use the name *previous* because ISQL always saves the current SQL statement, and names it *previous* when a new SQL statement is typed.

Protecting a Stored Statement

If the name you use with the STORE command is the name of a statement that is already stored, and you do *not* use the REPLACE keyword, you receive a warning message. The warning gives you the following choices:

- REPLACE the existing stored SQL statement with the new statement.
- END the processing of the STORE command, leaving the existing stored SQL statement intact.
- Enter a new name under which your new statement is to be stored.

The message itself appears as in Figure 73 on page 120.

```
ARI7955I The system ended your query result to process your command.
ARI7577D A stored SQL statement named DEPT already exists.
         Enter a new name to store the SQL statement, or enter
         one of the following keywords:
         REPLACE - to replace the existing stored SQL statement, or
         END - to end the store command processing.
```

Figure 73. Warning Message Displayed If You Try to Store a Name Already Stored

You can use the REPLACE function to replace an old statement with a new one. If you have already stored a statement under the name DEPT, you can still store the current statement under the same name by typing:

```
replace
```

and pressing ENTER.

If you do not want to replace the previously stored statement, simply type:

```
end
```

Press ENTER.

The END command returns you to the ISQL environment.

Starting a Stored Statement

When you start a stored SQL statement, ISQL moves it to the command buffer and then processes it. Start the statement stored as DEPT by typing the following:

```
start dept
```

When the results appear on your display, verify that the FORMAT commands you typed with the SELECT statement stored as DEPT are still in effect.

Note: Only the SELECT portion of the stored statement is recalled. The stored formatting commands are not shown although you can see their effect on the query.

Type END. The stored SELECT statement called DEPT is not erased. It remains in storage until you decide to erase it.

Starting a Stored Statement That Contains Placeholders

First, store a statement that contains placeholders. Type:

```
hold select &1 -
from &2 -
where &3

store myquery
```

Now start the statement, and add parameters to complete it. Type:

```
start myquery (* employee empno='000010')
```

This produces the following statement:

```
select * from employee where empno='000010'
```

and displays Figure 74 on page 121.

```
EMPNO   FIRSTNME      MIDINIT  LASTNAME        WORKDEPT  PHONENO  HIREDATE
------  ------------  -------  --------------  --------  -------  ----------
000010  CHRISTINE     I        HAAS            A00       3978     1965-01-01
* End of Result *** 1 Rows Displayed ***Cost Estimate is 1********************
```

Figure 74. A Query Result from Starting a Stored Statement with Placeholders

Note: A message is displayed, indicating that any formatting performed while viewing this query result is not saved. This condition is explained in detail in "Saving the Format Information."

Because the placeholders are replaced by the parameters when the statement is processed, the stored SQL statement remains unchanged and can be reused.

Recalling a Stored Statement

A stored SQL statement can be recalled to the command buffer to become the current SQL statement without being processed. You can do this to make changes to it and store it again, or to simply verify that it is the statement that you want to process next. For example, recall DEPT by typing:

```
recall dept
```

The recalled statement resembles:

```
SELECT DEPTNO,MGRNO FROM DEPARTMENT ORDER BY MGRNO
```

You can also use the RECALL command to display the current SQL statement, or to place the previous SQL statement in the command buffer at the top of the stack so that it becomes the current SQL statement. To recall the previous SQL statement, type:

```
recall previous
```

The statement that was current in the buffer becomes the new *previous* SQL statement.

To display the current SQL statement in the command buffer, type:

```
recall
```

or press PF5. The absence of a name on the RECALL command instructs ISQL to display the current SQL statement.

Saving the Format Information

The formatting information performed while viewing the query result is always saved when the query does not contain placeholders. When the query contains placeholders, however, the formatting information is saved only if the placeholders are not in the SELECT or FROM clause. To illustrate how format information is saved, remove the placeholders and insert actual values into the SELECT and FROM clauses of MYQUERY, but retain the placeholder in the WHERE clause by typing:

```
hold select * -
from employee -
where &1

store myquery replace
```

Notice the keyword REPLACE; it instructs ISQL that the statement you are now storing replaces the previous version of MYQUERY. If the keyword REPLACE is not used, ISQL issues the warning message described earlier. Of course, you can choose another name and save both the old and new queries.

Now start MYQUERY and include the parameter that completes the WHERE clause:

```
start myquery (empno='000010')
```

This processes the following statement:

```
select * -
from employee -
where empno='000010'
```

Add formatting information to the query result. Type:

```
format separator ' | '
```

End the query, and then save the formatting information by typing:

```
store myquery replace
```

Start the query again, but substitute a different value for the placeholder:

```
start myquery (empno='000150')
```

You see that the query result has retained the separator that you formatted and stored.

Changing a Stored Statement

Changes can be made to a stored SQL statement using the CHANGE command discussed earlier in this chapter under “Changing the Current SQL Statement” on page 90. For example, to change the ORDER BY clause in the statement stored as DEPT, perform the following:

1. Use the RECALL command to display the stored statement. Type:

   ```
   recall dept
   ```

   The recalled statement appears as:

   ```
   SELECT DEPTNO,MGRNO FROM DEPARTMENT ORDER BY MGRNO
   ```

2. For this example, change the ORDER BY feature to the DEPTNO column. Type:

   ```
   change /by mgrno/by deptno/
   ```

   The changed statement is displayed as:

   ```
   SELECT DEPTNO,MGRNO FROM DEPARTMENT ORDER BY DEPTNO
   ```

3. Now, store the changed SELECT statement as *DEPT2*:

   ```
   store dept2
   ```

4. To verify the new statement, use the RECALL command:

   recall dept2

If you look at the query results of DEPT and DEPT2, you will notice that the FORMAT information is maintained. As long as the statement to be changed or stored contains an unchanged SELECT or FROM clause, FORMAT information is saved.

Listing the Names of Stored Statements

To view the names and contents of stored statements, type:

list sql *

This displays information similar to Figure 75.

```
DEPT         SELECT DEPTNO,MGRNO FROM DEPARTMENT ORDER BY MGRNO
DEPT2        SELECT DEPTNO,MGRNO FROM DEPARTMENT ORDER BY DEPTN
MYQUERY      SELECT * FROM EMPLOYEE WHERE &1;
ARI7620I You have           3 stored SQL statements.
```

Figure 75. A Display of the Stored SQL Statements

Notice that only the first 50 characters of each statement are displayed.

You may wish to store a statement but are unsure if the stored name you want to use is already in use. Using the LIST SQL * command becomes tedious if there are many stored statements. Instead, you can list a specific stored statement by using its name in the LIST command. Type:

list sql getsup

You receive a message indicating that GETSUP is not found, and you can use it as the new store name.

You can also use this form of the LIST command to view statements whose name you know. For example, type the following to view DEPT2:

list sql dept2

You can use multiple statement names with the LIST command. For example, type:

list sql dept myquery

Renaming a Stored Statement

You can rename stored SQL statements. For example, type:

rename dept olddept

The stored statement DEPT is now called OLDDEPT. Do not use *previous* with the RENAME command.

Erasing a Stored Statement

When you no longer need a stored SQL statement, you can erase it. Type:

```
erase olddept
```

The stored statement called OLDDEPT has been erased.

You can erase several stored statements with a single ERASE command:

```
erase first second
```

EXERCISE 12 (Answers are in Appendix A, Answers to the Exercises, page 265.)

1. Recall the stored query MYQUERY.
2. Change the query so that information is retrieved for salaries between $25000 and $30000, and is ordered by the employee's first name.
3. Start the query.
4. Exclude the middle initial from the result.
5. Separate all columns with a blank, a vertical bar, an asterisk, a vertical bar, and a blank.
6. Change the EDLEVEL column heading to SCHOOL YEARS.
7. End the display and store the command along with all its formatting information using the name EXER11.
8. List all the stored SQL statements.
9. Retrieve the help information for the ISQL STORE command.

Chapter 9. Creating and Using Routines

A routine is a series of ISQL commands, SQL statements, or both, which is stored under an identifying name. When a routine is run, each command or statement in the routine is performed as if it had been typed from the keyboard. Routines save (and later run) frequently-used sequences of commands and statements, such as a series of ISQL SET commands that set certain operational characteristics.

Routines offer a number of features to users. In addition to using them to perform a frequently used set of commands or statements, you can:

- Define a profile routine that is run automatically when you start ISQL
- Use a routine to start one or more stored SQL statements
- Share routines with other users
- Use a routine to display and print query results.

Running Routines When ISQL Is Started

Profile Routines

A routine named *PROFILE*, that was previously set up by you, is performed automatically when you start ISQL. This allows unique terminal-session characteristics to be established before any information is typed. You can set these characteristics yourself by using the SET command described in Chapter 13, "ISQL Commands" on page 203.

DB2 Server for VM

Note: The PROFILE routines described in this section are ISQL PROFILE routines and should not be confused with a VM PROFILE EXEC.

Placeholders are not allowed in a profile routine. Because the routine is run automatically, there is no way to pass parameters to substitute for the placeholders.

In addition, a unique master profile routine can be created by someone having DBA authority. It is created with an authorization ID of SQLDBA and is run automatically for each user who starts ISQL as if it were the user's own profile routine.

Both the master and your own profile routines run automatically when you start ISQL. Your routine runs second, and its operational characteristics override those in the master routine.

Routines to Which Parameters Can Be Passed (DB2 Server for VM)

After the profile routine is run, you can invoke a subsequent routine to run as part of the ISQL signon process. Parameters can be passed on to this routine. The ISQL EXEC procedure invokes ISQL and runs this subsequent routine.

The ISQL EXEC procedure has the following format:

```
►►──ISQL──┬─────────┬──┬────────────────────────────────────────────────────────┬──────────────►◄
          └─V1SCRIO─┘  └─ROUTine──(──routine_name──┬──────────────────────┬──)──┘
                                                   └─(parameter_list)─────┘
```

You can create your own EXEC procedure from the ISQL EXEC procedure. For example, you can add commands to your EXEC to change the PF-key settings or to route printed reports.

You set your own terminal-session characteristics by using the SET command described in Chapter 13, “ISQL Commands” on page 203.

Using the ISQL Transaction Identifier (DB2 Server for VSE)

A profile routine cannot contain parameters, but an ISQL routine that runs as part of the ISQL signon procedure can contain parameters. You use the 4-character transaction identifier `ISQL` to start ISQL and to run a routine. Two ways of using the transaction identifier are:

1. Start ISQL, type your user ID and password, which the signon screen prompts you to do.
2. Start ISQL from another CICS transaction that suppresses the signon screen and the related terminal messages. The format for doing this along with its explanation are found in Appendix E, “Suppressing the ISQL Sign-On Display for DB2 Server for VSE” on page 287.

For method 1, the ISQL transaction identifier and routine have the following format:

```
►►──ISQL──┬──────────────────────────────────────────┬──────────────►◄
          └─routine_name──┬────────────────────┬─────┘
                          └─(parameter_list)───┘
```

Both of the above-mentioned methods let you specify a routine, and any parameters to be passed to that routine, to be run by ISQL after it has run a profile routine. For method 1, the ISQL transaction identifier and the routine name and all its parameters must not be longer than the screen width or 132, whichever is less. For method 2, the command which suppresses the ISQL signon display as described in Appendix E, “Suppressing the ISQL Sign-On Display for DB2 Server for VSE” on page 287, must not be longer than the screen width or 132, whichever is less.

You can insert any number of blanks after the routine name or the parameters and the surrounding parentheses. For example:

```
isql    routine  (abc (1 2   3))
```

You can create your own routine or profile routine. For example, in your profile, you can add commands to change the number of copies of printed reports or route printed reports.

You can change the terminal-session characteristics. You can do this each time you use ISQL (SET command, page 245), or you can have ISQL do it for you (SET commands in the PROFILE routine).

Establishing Where Routines Are Stored

Routines are stored in a special table called *ROUTINE*. It consists of four columns: NAME, SEQNO, COMMAND, and REMARKS (optional). An example of a routine table is shown in Figure 76.

Figure 76. Example of a Routine Table

NAME	SEQNO	COMMAND	REMARKS
EMPLREP	10	SELECT PROJNO,ACTNO,ACSTAFF -	GENERATE AND PRINT A
EMPLREP	20	FROM PROJ_ACT -	REPORT FOR PROJECT
EMPLREP	30	WHERE PROJNO IN (&1) -	STAFF
EMPLREP	40	ORDER BY PROJNO,ACTNO	
EMPLREP	50	FORMAT GROUP PROJNO	
EMPLREP	60	FORMAT SUBTOTAL ACSTAFF	
EMPLREP	70	FORMAT TTITLE 'AVERAGE PROJECT STAFF'	
EMPLREP	80	PRINT	
EMPLREP	90	END	
PRINTDEP	10	SELECT * FROM DEPARTMENT	FORMAT DEPARTMENT TABLE
PRINTDEP	20	FORMAT SEPARATOR ' | '	AND PRINT IT
PRINTDEP	30	FORMAT COLUMN DEPTNAME WIDTH 30	
PRINTDEP	40	PRINT	
PRINTDEP	50	END	

The NAME column identifies the rows that belong to a particular routine. SEQNO specifies the sequence in which the commands and statements are executed. Use sequence numbers that are increments of ten to allow for later additions. The COMMAND column contains the SQL statements and ISQL commands.

Before creating routines, you must have a routine table. You can create your own routine table if you have Resource authority or your own private dbspace (DB2 Server for VM user only). If you do not have Resource authority, ask the appropriate person to create a routine table for you. The following SQL statement illustrates the creation of a routine table:

```
create table routine (name char(8) not null, -
                      seqno integer not null, -
                      command varchar(254) not null, -
                      remarks varchar(254))
```

The CREATE TABLE statement is discussed in detail in Chapter 11, “Creating and Managing Tables” on page 175.

When creating this table, use the CREATE TABLE statement exactly as shown above, except for the REMARKS column which is optional. If it is included, specifying a data type of VARCHAR with a length of 40 is usually sufficient. The size selected for the REMARKS column should accommodate the largest entry used. Allow nulls so that remarks are not required. The COMMAND column can be a maximum length of 254 characters.

Once the table is created, create an index for it so that when the table is referenced, the commands or statements are displayed or executed in the correct order. To create an index, type a statement similar to the following (substitute a name of your choice for *RINDEX*):

```
create unique index rindex on routine -
               (name, seqno)
```

For detailed information on the CREATE INDEX statement, refer to Chapter 11, “Creating and Managing Tables” on page 175.

Storing a Routine

When you insert new commands or statements for a routine into the ROUTINE table, always assign the same routine name in the NAME column. Commands and statements are inserted into the routine table in the same manner as data is inserted into any database manager table by using SQL INSERT statements or the ISQL INPUT command.

The use of ampersands (&) in a routine is allowed only for creating placeholders (see the description of the RUN command for more information on placeholders).

In the COMMAND column of the routine, be sure to:

- Put single quotation marks around any placeholder that stands for a character data item. Enclose the COMMAND column (CHARACTER data type) in single quotation marks, and use a pair of single quotation marks for each single quotation mark that is to appear inside an SQL statement. This is shown in the example below.
- Use the continuation character as if you were typing a long SQL statement.

In the following example, the INPUT command inserts the commands and statements for a routine named *QREPORT* into the routine table.

```
input routine
'qreport',10,'select projno,acstaff -','begin:'
'qreport',20,'from proj_act -',null
'qreport',30,'where projno = '&1'' -',null
'qreport',40,'or actno = &2 -',null
'qreport',50,'order by projno',null
'qreport',60,'format group projno',null
'qreport',70,'format subtotal acstaff',null
'qreport',80,'print',null
'qreport',90,'end','done!'
 end
```

The stored information should resemble Figure 77.

Figure 77. Routine Statements Inserted in the Routine Table			
NAME	**SEQNO**	**COMMAND**	**REMARKS**
QREPORT	10	SELECT PROJNO,ACSTAFF -	BEGIN:
QREPORT	20	FROM PROJ_ACT -	
QREPORT	30	WHERE PROJNO = '&1' -	
QREPORT	40	OR ACTNO = &2 -	
QREPORT	50	ORDER BY PROJNO	
QREPORT	60	FORMAT GROUP PROJNO	
QREPORT	70	FORMAT SUBTOTAL ACSTAFF	
QREPORT	80	PRINT	
QREPORT	90	END	DONE!

To display your stored QREPORT routine, type:

```
select * -
from routine -
where name='qreport'
```

The display resembles Figure 78.

```
NAME             SEQNO  COMMAND                REMARKS
--------  ------------  ---------------------  --------------------
QREPORT             10  SELECT PROJNO,ACSTAF<  BEGIN:
QREPORT             20  FROM PROJ_ACT          ?
QREPORT             30  WHERE PROJNO = '&1'    ?
QREPORT             40  OR ACTNO = &2 -        ?
QREPORT             50  ORDER BY PROJNO        ?
QREPORT             60  FORMAT GROUP PROJNO    ?
QREPORT             70  FORMAT SUBTOTAL ACST<  ?
QREPORT             80  PRINT                  ?
QREPORT             90  END                    DONE!
* End of Result *** 9 Rows Displayed ***Cost Estimate is 1 ********
```

Figure 78. Display of Routine Statements Inserted in the Routine Table

Managing a Routine

Routines stored in a table are managed in the same manner as data in any table by using SQL UPDATE, INSERT, and DELETE statements.

For example, to modify the PRINTDEP routine (shown in Figure 76 on page 127) to use a double bar instead of a single bar to separate columns, you can type:

```
update routine -
set command = 'format separator  '' || ''' -
where seqno = 20 and name = 'printdep'
```

As another example, you can delete the entire QREPORT routine by typing:

```
delete from routine -
where name = 'qreport'
```

Running a Routine

You run routines by using the ISQL RUN command. Use placeholders and parameters in the same manner as you would use them in stored SQL statements.

For example, type the following command and placeholder replacement values to run the QREPORT routine you created:

```
run qreport ('ad3100' 180)
```

Routine QREPORT creates a printed report similar to that shown in Figure 79. The query result is printed rather than displayed because of the PRINT command contained in the routine. Producing a query result on the display is discussed later in this chapter under “Using SELECT Statements in a Routine” on page 131.

```
.                                                                    .
. 08/27/89   SELECT PROJNO,ACSTAFF FROM PROJ_ACT WHERE PROJNO = 'AD3    .
.                                                                    .
.                                                                    .
.                                                                    .
. PROJNO  ACSTAFF                                                    .
. ______  _______                                                    .
. AD3100     0.50                                                    .
.         _______                                                    .
. ******     0.50                                                    .
.                                                                    .
. AD3111     1.00                                                    .
.         _______                                                    .
. ******     1.00                                                    .
.                                                                    .
. AD3112     0.50                                                    .
.         _______                                                    .
. ******     0.50                                                    .
.                                                                    .
. AD3113     0.75                                                    .
.            1.00                                                    .
.            0.50                                                    .
.         _______                                                    .
. ******     2.25                                                    .
.                                                                    .
. MA2112     1.00                                                    .
.            1.00                                                    .
.                                                                    .
```

Figure 79. Example of a Report Created from a Routine

Running Shared Routines

You can run another user’s routine if you have obtained the SELECT privilege (using a GRANT statement as described in “Sharing Your Tables with Other Users” on page 186) on that user’s routine table.

Error Mode Processing in a Routine

When an error is detected in a routine, continued processing depends on a setting you make to a special RUNMODE indicator. It indicates the mode of operation for running routines.

The format for the command that sets the indicator is:

```
>>──SET──RUNMode──┬─CONTInue─┬──────────────────────────────────────────><
                  ├─STOP─────┤
                  └─CANCEL───┘
```

CONTInue
: indicates that your routine continues to the next command or statement even if errors are detected. You are likely to use this option when your routine contains several independent commands or statements. Failure of a particular command or statement does not affect the remaining commands and statements.

  If you do not specify a RUNMODE option, CONTINUE becomes the default value.

STOP
: indicates that your routine is ended, but a ROLLBACK operation is not performed. Select this option when commands and statements within a routine are not interrelated, and you want to maintain any changes made to the database. The remaining commands and statements are not executed if errors exist in any of their predecessors.

CANCEL
: indicates that your routine is ended and a ROLLBACK operation is performed. Select this option when your routine contains a series of interrelated commands and statements that update tables. When an error is detected, all changes generated by the routine are erased and the integrity of the database is maintained.

You can display the current RUNMODE setting by issuing the following command:

```
list set runmode
```

You can set the RUNMODE indicator at any time, either from the display terminal or in a routine.

Using INPUT Commands in a Routine

If the ISQL INPUT command is used in a routine, all data and INPUT subcommands (SAVE and BACKOUT) must also be entered from the routine.

Using SELECT Statements in a Routine

The use of SELECT statements in a routine (either an actual statement contained in the routine or a stored statement started by the routine) is slightly different from usage of the statements at a keyboard. SELECT statements contained in a routine do not cause query results to be displayed automatically at the terminal.

The SELECT statement results can be displayed at the terminal by placing an ISQL DISPLAY command in the routine at the desired location. This command can be placed anywhere between the SELECT statement and its associated END command. The DISPLAY command allows results to be formatted before they are displayed. It also lets you type display commands from the keyboard after the routine results are displayed.

To illustrate this process, type the following command:

```
insert into routine (name,seqno,command) -
values ('qreport',75,'display')
```

The QREPORT routine now appears as Figure 80.

Figure 80. Modified QREPORT Routine

NAME	SEQNO	COMMAND	REMARKS
QREPORT	10	SELECT PROJNO, ACSTAFF -	BEGIN:
QREPORT	20	FROM PROJ_ACT -	
QREPORT	30	WHERE PROJNO = '&1' -	
QREPORT	40	OR ACTNO = &2 -	
QREPORT	50	ORDER BY PROJNO	
QREPORT	60	FORMAT GROUP PROJNO	
QREPORT	70	FORMAT SUBTOTAL ACSTAFF	
QREPORT	75	DISPLAY	
QREPORT	80	PRINT	
QREPORT	90	END	DONE!

Run the routine by typing:

```
run qreport ('ad3100' 180)
```

The formatted query result is now displayed at your terminal. If required, you can alter it using additional FORMAT commands. The display resembles Figure 81.

```
PROJNO  ACSTAFF
------  -------
AD3100     0.50
        -------
******     0.50

AD3111     1.00
        -------
******     1.00

AD3112     0.50
        -------
******     0.50

AD3113     0.75
           1.00
           0.50
        -------
******     2.25

MA2112     1.00
           1.00
        -------
******     2.00
```

Figure 81. Formatted Query Result from Running Routine QREPORT

Type an END command to end the display of the query result and return to the routine. The routine issues its remaining commands, causing a report containing the modified query results to be printed.

EXERCISE 13 (Answers are in Appendix A, Answers to the Exercises, page 266.)

1. Create a routine named EXER13 to:

   a. Select the ACTNO and ACTDESC columns from the ACTIVITY table.

   b. Provide a separation between columns consisting of three blanks.

   c. Display the results on the screen.

   d. Request three copies of the resulting report.

   e. End the query.

   Note: Remember to take the REMARKS column of the ROUTINE table into account if you have such a column.

Chapter 10. Using Additional Query Techniques

This chapter builds on information in the previous chapters. The new query techniques described in this chapter fall into these major categories:

- Selecting data from two or more tables
- Using correlation names
- Using groups to get summary information
- Working with subqueries
- Working with views
- Combining multiple queries
- Canceling running commands.

Selecting Data from Two or More Tables

All queries discussed so far have used only one table, which you specified in the FROM clause of the query.

With the *join* feature of the database manager, you can write a query that combines two or more tables or views. (The use of join on views is discussed later in this chapter.) The maximum number of table or view references is 15; none of the tables or views can have the same name.

To perform a table join, list all the tables involved in the join operation in the FROM clause and separate them by commas. This enables you to select data from each table and combine it into a single query result. Suppose you want the surnames of the employees involved in project IF1000. This information is contained in the EMPLOYEE and EMP_ACT tables. Type:

```
select projno,lastname -
from employee,emp_act -
where employee.empno = emp_act.empno -
and projno = 'IF1000' -
order by projno,lastname
```

This displays Figure 82.

```
PROJNO  LASTNAME
------  ---------------
IF1000  KWAN
IF1000  NICHOLLS
IF1000  QUINTANA
IF1000  QUINTANA
* End of Result *** 4 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 82. A Query Result Displaying Data from Two Tables

Notice that some of the column names in the statement are prefixed with either EMPLOYEE or EMP_ACT, while others are not. This prefix differentiates columns which have the same name but are contained in different tables. Column names that are unique among the tables in the join operation do not require such a prefix.

The WHERE clause specifies which data in the tables is to be selected:

```
where employee.empno = emp_act.empno -
```

The condition in the WHERE clause that specifies this relationship between two tables is a *join condition*. A join condition is never satisfied by a null value. If a row in any table involved in a join condition contains a null value, the row is ignored.

If no join condition is given in the WHERE clause, *all possible combinations* of rows from the tables named in the FROM clause are displayed. This is normally not the desired result, so you should always consider adding a join condition when connecting tables.

If *SELECT ** is used in a join query, all columns of the first table or view, followed by all the columns of the second table or view, are selected.

If a join query uses a qualified column name in its SELECT clause, the same qualified column name can be used in the ORDER BY clause. For example, ORDER BY X.EMPLOYEE and ORDER BY EMP_ACT.EMPLOYEE are acceptable.

If a field procedure is defined on a column in the ORDER BY clause, it affects the order of the rows. See the following manuals for more information about field procedures:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VM System Administration*
- *DB2 Server for VSE Application Programming*
- *DB2 Server for VM Application Programming*

A view whose definition involves a GROUP BY cannot be joined with another table or view.

Joining a Table to Itself

It is possible to write a query in which a table is joined to itself. You use the table name two or more times in the FROM clause.

Using the table name more than once indicates that the join consists of combinations of two or more rows from the same table. Because the table name is no longer unique, each table name in the FROM clause must be given a unique *correlation name*. The correlation name can be any string of as many as 18 characters, beginning with a letter. The correlation names identify column names in the SELECT and WHERE clauses.

For example, you want the total of the values from the ACSTAFF column (PROJ_ACT table) for activities 60 and 70 for any project that contains both these activities. Type:

```
select pa1.projno,pa1.acstaff + pa2.acstaff -
from proj_act pa1,proj_act pa2 -
where pa1.projno = pa2.projno -
and pa1.actno = 60 and pa2.actno = 70 -
order by 1
```

This provides a result similar to Figure 83 on page 137.

```
PROJNO    EXPRESSION 1
------  --------------
AD3111            2.30
AD3111            1.30
AD3111            2.00
AD3111            1.00
AD3112            1.50
AD3112            1.25
AD3112            1.75
AD3112            1.00
AD3112            1.25
AD3112            1.00
AD3112            1.50
AD3112            0.75
AD3112            1.50
AD3112            1.25
AD3112            1.75
AD3112            1.00
AD3112            1.75
AD3112            1.50
AD3112            2.00
AD3112            1.25
AD3113            1.25
AD3113            1.75
```

Figure 83. A Query Result Where the Table is Joined to Itself

In this query, the PROJ_ACT table is treated as if it is two different tables, which are named *PA1* and *PA2*. In PA1, the database manager searches for rows with activity 60 (`PA1.ACTNO = 60`). In PA2, it searches for rows with activity 70 (`PA2.ACTNO = 70`). The information from a PA1 row is combined with the information from a PA2 row with the same project number (`PA1.PROJNO = PA2.PROJNO`).

Sorting in a Join

If a sort is needed because of the join, consideration must be given to the length of the encoded key that is built by the sort. The length of the encoded key must not exceed 255 bytes. It is the sum of the lengths of the columns, plus approximately 25% of the lengths of any columns that are of varying-length character type.

Qualifying Table Names with Their Owner

A table name must be qualified by an owner if the table was created by another user. For example, to access the EMPLOYEE table created by SQLDBA, refer to this table as SQLDBA.EMPLOYEE.

The table name qualifying the column must include the owner if the table name in a FROM clause is qualified by an owner. This table name qualifies a column reference in a WHERE clause, ORDER BY clause, GROUP BY clause, or HAVING clause. For example, the following query accurately identifies both tables and columns:

```
select projname from sqldba.project,sqldba.proj_act -
   where sqldba.project.projno = sqldba.proj_act.projno
```

If you are the creator of the tables, this query could be typed as:

```
   select projname from project,proj_act -
   where project.projno = proj_act.projno
```

Using Correlation Names in Queries

Queries become rather wordy when a column name is qualified by its table name. To simplify such a query, you can define a correlation name for a table. For example, the query in “Selecting Data from Two or More Tables” on page 135 can be typed as:

```
select projno,lastname -
from employee e,emp_act ea -
where e.empno = ea.empno -
and projno = 'IF1000' -
order by projno,lastname
```

In this example, the letter *e* refers to the EMPLOYEE table and *ea* refers to the EMP_ACT table. Correlation names are defined in the FROM clause by placing the correlation name after the table name to which it refers. A blank must separate the table name from its correlation name. Correlation names must begin with a letter and can be up to 18 characters long.

Correlation names must not duplicate any other correlation name or table name in the query. In the FROM clause, the tables cannot have identical names as well as identical correlation names. If you use duplicate names, you receive an error message.

If a column name is qualified with a table name or correlation name, the column name must be within the scope of the table name or correlation name.

The following example is valid because the DEPTNO column name is qualified with the PROJECT table name.

```
select project.deptno from project
```

The following example is also valid, but this time because the DEPTNO column name is qualified with the PRO correlation name.

```
select pro.deptno from project pro
```

Using Correlation Names to Combine Information in the Same Table

You can use a correlation name for more than just a means of abbreviating a table name in a query. It also lets you reference a single table as though it were two (or more) different tables. In this case, you can list the same table twice in the FROM clause, but you must give each table name a unique correlation name.

For example, assume you want the combined estimated mean number of employees needed for activities 60 and 70 for any project that contains these two activities. Type:

```
select p1.projno,p1.acstaff + p2.acstaff -
from proj_act p1,proj_act p2 -
where p1.projno = p2.projno -
and p1.actno = 60 and p2.actno = 70 -
order by 1
```

This displays Figure 84 on page 139.

```
PROJNO   EXPRESSION 1
------  --------------
AD3111            2.30
AD3111            1.30
AD3111            2.00
AD3111            1.00
AD3112            1.50
AD3112            1.25
AD3112            1.75
AD3112            1.00
AD3112            1.25
AD3112            1.00
AD3112            1.50
AD3112            0.75
AD3112            1.50
AD3112            1.25
AD3112            1.75
AD3112            1.00
AD3112            1.75
AD3112            1.50
AD3112            2.00
AD3112            1.25
AD3113            1.25
AD3113            1.75
```

Figure 84. A Query Result from Using Correlation Names

Correlation names must not duplicate each other or any of the table names in the query.

Selecting Summary Information by Groups

In "Using Column Functions" on page 45, you learned how to select information using the column functions AVG, SUM, MIN, MAX, and COUNT. It was pointed out that a column function returns a single value for the query result. Sometimes it is necessary to perform several queries containing a column function to solve a problem. For example, to find the average estimated mean number of employees for each project in the PROJ_ACT table you can type:

```
select avg(acstaff) -
from proj_act -
where projno = 'AD3100'
  .
  .
  .
select avg(acstaff) -
from proj_act -
where projno = 'AD3111'
  .
  .
  .
```

This procedure can also be performed using a single query that contains a GROUP BY clause:

```
select projno,avg(acstaff) -
from proj_act -
group by projno -
order by projno
```

This displays Figure 85 on page 140.

```
 PROJNO                       AVG(ACSTAFF)
 ------  ---------------------------------
 AD3100       0.500000000000000000000000
 AD3110       1.000000000000000000000000
 AD3111       0.935714285714285714285714
 AD3112       0.622727272727272727272727
 AD3113       0.846153846153846153846153
 IF1000       0.600000000000000000000000
 IF2000       0.550000000000000000000000
 MA2100       0.750000000000000000000000
 MA2110       0.750000000000000000000000
 MA2111       1.000000000000000000000000
 MA2112       1.214285714285714285714285
 MA2113       1.071428571428571428571428
 OP1000       0.250000000000000000000000
 OP1010       2.500000000000000000000000
 OP2000       0.750000000000000000000000
 OP2010       1.000000000000000000000000
 OP2011       0.500000000000000000000000
 OP2012       0.500000000000000000000000
 OP2013       0.500000000000000000000000
 PL2100       0.833333333333333333333333
 * End of Result ************* 20 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 85. A Query Result Displaying Summary Information by Group

In the above example, the database manager performs grouping as follows:

1. A project number is selected from the PROJ_ACT table (using the SELECT clause).
2. All the rows that have the same project number are grouped (using the GROUP BY clause), and the average mean number of employees is calculated. This produces one row of the query result.
3. Another project number is selected, grouped, the average mean is calculated, and another row for the query result is produced. This process continues until all project numbers have been selected.

Note: All rows containing a null value (indicated by a question mark) in the grouping column are grouped together, just as they would be if they contained identical numeric or character values in that column. All rows containing arithmetic errors (indicated by number signs) in the grouping column are grouped together and appear after the null values.

When used, the GROUP BY clause always follows the WHERE clause (or the FROM clause if no WHERE clause is in the command). It identifies which selected column to use for grouping results. All columns included in the SELECT clause must either have an associated column function or also appear in the GROUP BY clause.

Specifying a Search Condition for Groups

You may want to select only *grouped* summary information. For example, to find the average mean number of employees for projects having more than three activities, type:

```
select projno,avg(acstaff) -
from proj_act -
group by projno -
having count(*) > 3 -
order by projno
```

This displays Figure 86.

```
PROJNO                        AVG(ACSTAFF)
------  ---------------------------------
AD3111       0.935714285714285714285714 2
AD3112       0.622727272727272727272727 2
AD3113       0.846153846153846153846153 8
IF1000       0.600000000000000000000000 0
IF2000       0.550000000000000000000000 0
MA2111       1.000000000000000000000000 0
MA2112       1.214285714285714285714285 7
MA2113       1.071428571428571428571428 5
* End of Result *** 8 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 86. A Query Result of Grouped Summary Information

You can think of the HAVING clause as the WHERE clause for groups. When used, the HAVING clause always comes after the GROUP BY clause. It states a search condition to be applied to each group. If a group satisfies this condition, it is included in the query result.

The search condition must qualify the group. For example, the condition can be the result of a column function such as COUNT, SUM, or AVG. The condition cannot apply to individual rows within the group.

If you are grouping values in a VARCHAR column, the GROUP BY clause puts together all the values that differ only by trailing blanks. Therefore, 'Adm services' and 'Adm services ' are grouped together.

The HAVING clause is always evaluated *before* the SELECT clause in a SELECT statement to eliminate groups of rows that would otherwise cause arithmetic errors during evaluation of the SELECT clause.

Consider, for example, the following query on the EMPLOYEE table:

```
select workdept,sum(salary) / sum(comm) -
from employee -
group by workdept -
having sum(comm) <> 0 -
order by workdept
```

For those departments that do not get a commission, the sum of commissions would add up to zero. If this were the case, and no commission group totals were qualified in a HAVING clause, the database manager could not evaluate the expression SUM(SALARY) / SUM(COMM), because division by zero is an invalid arithmetic operation. The problem is solved by the HAVING clause, which specifies only those groups that do *not* add up to zero commission. Because the HAVING clause is evaluated *before* the SELECT clause, all groups that would otherwise have a total commission of zero are eliminated before any rows are selected from the table by the SELECT clause.

Using Subqueries to Build Search Conditions

Suppose you want to view the salaries of all employees assigned to project OP1010. First you query the EMP_ACT table to determine the employees assigned to that project; then you query the EMPLOYEE table for these employees to view their salaries. The first query can be made a *subquery* of the second, and the entire process can be performed in one statement by typing:

```
select empno,salary -
   from employee -
   where empno in -
   (select empno -
    from emp_act -
    where projno = 'OP1010') -
    order by empno
```

This produces the display in Figure 87.

```
EMPNO        SALARY
------  -----------
000090     29750.00
000280     26250.00
000290     15340.00
000300     17750.00
000310     15900.00
* End of Result ************** 5 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 87. A Query Result from Using a Subquery

Subqueries must be placed last in a WHERE clause or HAVING clause. They cannot be used with the BETWEEN, LIKE, or NULL predicates.

When a subquery is used in a comparison, the subquery must contain only a single column or expression in its SELECT clause. It can also have a GROUP BY and HAVING clause. The subquery returns only one value except when using the IN predicate or when modifying the comparison operator with ALL, ANY, or SOME. (Modified comparison operators are discussed later in this chapter).

For example, the following query finds the activities for project IF1000 whose estimated mean number of employees is greater than the minimum estimated mean for that project:

```
select actno,acstaff -
from proj_act -
where projno = 'IF1000' -
  and acstaff > -
       (select min(acstaff) -
        from proj_act -
        where projno = 'IF1000') -
order by actno
```

Used with the IN Predicate

The IN predicate links to a subquery that returns a set of values. For example, the following query lists the surnames of employees responsible for projects MA2100 and OP2012:

```
select lastname -
from employee -
where empno in -
       (select respemp -
        from project -
        where projno = 'MA2100' -
        or projno = 'OP2012')
```

The subquery is evaluated once, and the resulting list is substituted directly into the outer-level query. For example, if the subquery above selects employee numbers 60 and 330, the outer-level query is evaluated as if its WHERE clause is WHERE EMPNO IN (60, 330).

Used in UPDATE Statements

Subqueries used in UPDATE statements cannot be based on the same table in which rows are to be updated. The following example illustrates an UPDATE statement with a subquery. It adds five days to the estimated completion date of any activity for project IF1000 whose activity keyword begins with *A*:

```
update proj_act -
set acendate = acendate + 5 days -
where projno = 'IF1000' -
  and actno in -
       (select actno -
        from activity -
        where actkwd like 'A%')
```

Used in DELETE Statements

Subqueries used in DELETE statements have the following restrictions, where T1 denotes a table that is referenced in the FROM clause of the subquery, T2 denotes the object table of a DELETE statement, and T3 denotes another table that is delete-connected to T2:

- T1 and T2 must not be the same table.
- T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL.
- T1 must not be a dependent of T3 in a relationship with a delete rule of CASCADE or SET NULL, if deletions from T2 cascade to T3.

The following example illustrates a DELETE statement with a subquery. It deletes all activities for any project whose major project is OP1000.

```
delete from emp_act -
where projno = -
       (select projno -
        from project -
        where majproj = 'OP1000')
```

Uses for Subqueries

A subquery can include a join, a grouping, or one or more lower-level subqueries. Many subqueries can be included in the same outer-level query, each contained in its own search condition and enclosed in parentheses. The following example shows how a join and a subquery might be combined to solve a problem. The query lists the department and project names of departments with projects having an activity with a start date of January 1, 1983:

```
select deptname,projname -
from department,project -
where department.deptno = project.deptno -
  and projno in -
       (select projno -
        from proj_act -
        where acstdate = '1983-01-01')
```

You are not limited to only one subquery, and you are not limited to using subqueries only to calculate a list. Up to 16 subqueries can be used and any of the operators (for example, =, ¬=, >, and <>) can be used to link them to the next higher-level query. For example, assume you want the names of the projects associated with the activity whose keyword is *TEACH*. Type:

```
select projname -
from project -
where projno in (select projno -
                 from proj_act -
                 where actno = (select actno -
                               from activity -
                               where actkwd = 'TEACH'))
```

The operators you use to link the subqueries can be modified by ALL, ANY, or SOME.

Modified Comparison Operators

When used with a subquery, the comparison operators =, <>, >, >=, <, and <= can be *modified* by ALL , ANY, or SOME. These words permit the subquery to return a set of values. They also determine how the set is to be treated in the search-condition where it appears. Using the comparison operator > as an example (the remarks below apply to the other operators as well), ALL and ANY are used like this:

field > **(subquery)**
: denotes that the subquery must return exactly one value; otherwise, an error condition results. The condition is satisfied if the given field is greater than the value returned by the subquery.

field > **ALL (subquery)**
: denotes that the subquery may return a set of values. The condition is satisfied if the given field is greater than each individual value in the returned set. If the subquery returns an empty set, the condition is satisfied.

field > **ANY (subquery)**
: denotes that the subquery may return a set of values. The condition is satisfied if the given field is greater than at least one of the values in the set. If the subquery returns an empty set, the condition is not satisfied.

SOME can be used as a synonym for ANY.

The following example uses an ALL comparison to find those projects with activities whose estimated mean number of employees is greater than all of the estimated mean number of employees for activity 10 for project AD3100:

```
select projno,actno -
from proj_act -
where acstaff > all -
        (select acstaff -
         from proj_act -
         where actno = 10 -
         and projno = 'AD3100')
```

Unmodified Comparison Operators

If a subquery is linked to an outer-level query by an *unmodified* comparison operator such as = or >, the subquery cannot contain a GROUP BY or HAVING clause. However, a subquery can contain a GROUP BY or HAVING clause if it is used with a comparison operator modified by ALL or ANY, or by an IN or EXISTS predicate. The EXISTS predicate is described under "Testing for Existence" on page 159.

The subquery capability is quite powerful and can also be used in UPDATE and DELETE statements. The GROUP BY and HAVING clauses can also be used in subqueries.

Using a Correlated Subquery to Build a Search Condition

There may be times when you need to relate the search condition in a subquery to a value in each row of the table named in the main query. For example, suppose you want to know the project and activity numbers for activities that have an estimated mean number of employees that is less than the average estimated mean for that activity. First you need a main query to select rows from the PROJ_ACT table:

```
select projno,actno,acstaff -
from proj_act -
where acstaff < ...
```

Second you need a subquery that calculates the average estimated mean for each activity being considered for selection in the main query:

```
 .
 .
 .
where acstaff < (select avg(acstaff) -
       from proj_act -
       where actno = ...)
```

This subquery, which calculates the average mean, cannot do so until the activity number that is being considered for selection in the main query is provided. A method is needed to correlate the activity number being considered for selection in the main query with the search condition in the subquery. This association is made with a correlation name as shown in the finished query:

```
select projno,actno,acstaff -
from proj_act x -
where acstaff < (select avg(acstaff) -
          from proj_act -
          where actno = x.actno)
```

In this example, x is the correlation name that links the two queries. It enables a new average to be calculated for each activity being considered for selection in the main query. This differs from the ordinary subquery, which evaluates the subquery only once and then uses the single result in the main query. The correlation name that you choose must be different from any column name or correlation name in the main query and subquery.

As with subqueries, correlated subqueries can also be applied to UPDATE and DELETE statements.

The correlation name must be defined in the FROM clause of a query that contains the correlated subquery. However, this containment may involve several levels of nesting, as illustrated in the following example. It finds employees currently assigned to and responsible for, at least two projects. Note how the outer-level query is correlated to a second-level subquery:

```
select empno,lastname -
from employee y  -
where 2 <= -
  (select count(distinct projno) -
   from emp_act -
   where empno = y.empno -
     and projno in -
        (select projno -
         from project -
         where respemp = y.empno))
```

Used in a Join Query

The outer-level query that defines a correlation name can be a join query. In this case, it has more than one table name in its FROM clause, and the correlation name is associated with one of these table names. The following example lists projects with activities whose estimated mean number of employees is less than the average mean. In this example, the project name is also selected by means of a join:

```
select project.projno,projname,actno,acstaff -
from project,proj_act x -
where project.projno = proj_act.projno -
  and acstaff < -
    (select avg(acstaff) -
     from proj_act -
     where actno = x.actno) -
order by 1,3
```

In the above example, the correlated subquery is used in the WHERE clause and is evaluated once per row of the outer-level query.

Used in a HAVING Clause

A correlated subquery can also be used in a HAVING clause, in which case it is evaluated once per *group* of the outer-level query as defined by its GROUP BY clause. If a subquery is used in a HAVING clause, the correlated column reference in the subquery must be a property of each group. It must be either the columns you want to group by, or some other column used with a function.

The following statement uses a correlated subquery in the HAVING clause.

```
select projno -
from proj_act p -
group by projno -
having sum(acstaff) &gltsym.= (select avg(acstaff) -
                        from proj_act -
                        where projno = p.projno)
```

If a table name in a FROM clause is qualified by an owner, and this table name qualifies a column reference in a WHERE, ORDER BY, GROUP BY, or HAVING clause, the column-reference qualifier must also include the owner, along with the table name.

Correlated Function

The use of a function with a correlation name in a subquery is called a *correlated function*. The correlated function must contain one correlated column that is not an expression (for example, X.SALARY). A correlated function can specify the DISTINCT option, such as COUNT(DISTINCT X.DEPTNO)). If so, the DISTINCT counts as the single permitted DISTINCT specification for the outer-level query (remember that a query can use DISTINCT only once).

Used in DELETE or UPDATE Statements

As with subqueries, correlated subqueries can also be applied to UPDATE and DELETE statements. The following example illustrates an UPDATE statement with a correlated subquery that uses correlation name *ABC*. It writes the word *INACTIVE* in the PROJNAME column of those projects that do not have an entry in the PROJ_ACT table.

```
update project abc -
set projname = 'inactive' -
where not exists -
  (select * -
  from proj_act -
  where projno = abc.projno)
```

The following example shows the use of a correlated subquery with a DELETE statement. The statement would delete those employees from the EMPLOYEE table who have no entries in the EMP_ACT table:

```
delete from employee x -
where 0 = -
  (select count(*) -
  from emp_act -
  where empno = x.empno)
```

A correlated subquery inside a DELETE statement has the following restrictions, where T1 denotes a table that is referenced in the FROM clause of the subquery statement, T2 denotes the object table of a DELETE statement, and T3 denotes another table that is delete-connected to T2:

- T1 and T2 must not be the same table.
- T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL.
- T1 must not be a dependent of T3 in a relationship with a delete rule of CASCADE or SET NULL, if deletions from T2 cascade to T3.

Using Views to Simplify Queries

A lengthy query may just be the beginning of your questions. Having found an answer to your query, you might want to enter additional queries against that answer. Instead of storing the query statement, you can access a simple table that holds the results of the query. You do this by using a database manager *view*.

A view is an imaginary table that logically contains the results of a query. Because the results are tabular, the view resembles a database manager table and can be queried using SQL SELECT commands as though it were an actual table.

The following example creates a view called *ACT10*. It selects information for projects that contain activity 10:

```
create view act10 -
as select p.projno,projname,acstaff -
   from project p,proj_act pa -
   where p.projno = pa.projno -
      and actno = 10
```

Type the following statement to display this view:

```
select * -
from act10 -
order by projno
```

This query presents the display in Figure 88.

```
PROJNO  PROJNAME                  ACSTAFF
------  ------------------------  -------
AD3100  ADMIN SERVICES               0.50
AD3110  GENERAL AD SYSTEMS           1.00
IF1000  QUERY SERVICES               0.50
IF1000  QUERY SERVICES               0.50
IF2000  USER EDUCATION               0.50
MA2100  WELD LINE AUTOMATION         0.50
MA2110  W L PROGRAMMING              1.00
OP1000  OPERATION SUPPORT            0.25
OP1010  OPERATION                    1.00
OP2010  SYSTEMS SUPPORT              1.00
* End of Result ************* 10 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 88. A Query Result Showing the Contents of the ACT10 View

When defining a view, you cannot use an ORDER BY clause. Ordering is specified when querying the view.

Views are not limited to tables; they can be written on other views as well. You can use almost any kind of query against almost any kind of view. Techniques such as joining, grouping, and subquerying can be combined arbitrarily.

Naming Columns in a View

Sometimes it is necessary to provide new column names for a view. For example, suppose in the view just created you want .10 subtracted from the estimated mean number of employees in any row selected from the PROJ_ACT table. A meaningful column title for the subtraction result, such as *REDUCED_MEAN*, can be created by using the CREATE VIEW statement. To illustrate, create a view similar to the view you just created, using 60 as the activity number. Type:

```
create view act60 (projno,projname,reduced_mean) -
as select p.projno,projname,acstaff-.10 -
   from project p, proj_act pa -
   where p.projno = pa.projno -
      and actno = 60
```

Now type the following statement to display this view:

```
select projno,projname,reduced_mean -
from act60 -
order by projno
```

This query presents Figure 89.

```
PROJNO  PROJNAME                  REDUCED_MEAN
------  ------------------------  ------------
AD3111  PAYROLL PROGRAMMING               0.70
AD3111  PAYROLL PROGRAMMING               0.40
AD3112  PERSONNEL PROGRAMMG               0.65
AD3112  PERSONNEL PROGRAMMG               0.40
AD3112  PERSONNEL PROGRAMMG               0.65
AD3112  PERSONNEL PROGRAMMG               0.90
AD3113  ACCOUNT PROGRAMMING               0.65
AD3113  ACCOUNT PROGRAMMING               0.90
AD3113  ACCOUNT PROGRAMMING               0.15
MA2111  W L PROGRAM DESIGN                0.90
MA2111  W L PROGRAM DESIGN                0.90
MA2112  W L ROBOT DESIGN                  1.90
MA2113  W L PROD CONT PROGS               0.90
MA2113  W L PROD CONT PROGS               0.90
* End of Result ************* 14 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 89. A Query Result Showing the Named Columns of a View

You must provide column names for any views that contain a column whose values are calculated by an expression or a column function.

Views are discussed in several sections of this manual. For more information, see “Using a View to Restrict Privileges to Certain Rows” on page 190 and “Using a View to Restrict Privileges to Certain Columns” on page 190.

EXERCISE 14 (Answers are in Appendix A, Answers to the Exercises, on page 266.)

Perform the following:

1. Retrieve the activity number, employee number, project number, employee time, and activity description for all employees in the EMP_ACT table. Order the results by activity number.
2. Retrieve the total employee time for each activity. Order the results by activity number.
3. Create a view called EMPLS that has the following columns from the EMP_ACT and EMPLOYEE tables:

```
ACTNO    (rename ACTIVITY)
EMPTIME  (rename FRACTION_TIME)
LASTNAME (rename SURNAME)
```

4. Using the view created in the above step, retrieve the activity number, proportion of employee time, and last name for activity 100.
5. Using the view EMPLS, select the entire view. Order the results by activity number and proportion of employee time.
6. Create an outline format for the ACTIVITY column of EMPLS and create a top title for this report called WHO IS DOING WHAT.
7. Retrieve information in the PROJ_ACT table for activities that are currently performed for project number AD3113. Order the results by activity number.
8. Find the activity number and activity staff for activities ending on January 1, 1983, where the activity number is greater than the maximum activity number for project IF2000.
9. Retrieve the last name and salary of all the designers whose salary is greater than the average employee salary.

Using a Correlated Subquery with a Join

As you recall from working with subqueries and joins earlier in this chapter, subqueries produce lists to be used in the main query. You can also correlate a subquery to the main query by using a correlation name. The following query contains both a correlated subquery and a join:

```
select mx.projno,max(mx.emptime),mx.empno, -
min(mn.emptime),mn.empno -
from emp_act mn,emp_act mx -
where mx.projno = mn.projno -
and mx.emptime = (select max(emptime) -
           from emp_act -
           where projno = mx.projno) -
and mn.emptime = (select min(emptime) -
           from emp_act -
           where projno = mn.projno) -
group by mx.projno,mx.empno,mn.empno -
order by mx.projno
```

This query finds the maximum and minimum employee time proportions for each project in the EMP_ACT table, along with the associated employee numbers for those maximum and minimum values. The result is displayed in Figure 90 on page 151.

```
PROJNO    MAX(EMPTIME)  EMPNO     MIN(EMPTIME)  EMPNO
------  --------------  ------  --------------  ------
AD3100            0.50  000010            0.50  000010
AD3110            1.00  000070            1.00  000070
AD3111            1.00  000240            0.50  000230
AD3111            1.00  000230            0.50  000230
AD3112            1.00  000250            0.25  000250
AD3113            1.00  000260            0.25  000270
AD3113            1.00  000270            0.25  000270
IF1000            1.00  000130            0.50  000030
IF1000            1.00  000130            0.50  000130
IF1000            1.00  000130            0.50  000140
IF2000            1.00  000140            0.50  000030
IF2000            1.00  000140            0.50  000130
IF2000            1.00  000140            0.50  000140
MA2100            1.00  000110            0.50  000010
MA2110            1.00  000010            1.00  000010
MA2111            1.00  000200            1.00  000200
MA2111            1.00  000200            1.00  000220
MA2111            1.00  000220            1.00  000200
MA2111            1.00  000220            1.00  000220
MA2112            1.00  000150            1.00  000150
MA2112            1.00  000150            1.00  000170
MA2112            1.00  000150            1.00  000190
```

Figure 90. A Query Result from Using a Correlated Subquery and a Join

Remember, a join operation uses the values from two or more *tables* in the same query. To combine two or more *queries* into a single query result, use the UNION operator described in the next section.

Combining Multiple Queries

Two operators are used when combining multiple queries: UNION and UNION ALL.

Using the UNION Operator

The UNION operator provides a means of combining two or more queries into a single query by merging values found in one or more tables.

The format of the union operator is:

```
►►──UNION──┬─────┬──────────────────────────────────────────────►
           └─ALL─┘
```

Each of the queries connected by UNION is executed to produce an answer set. The UNION keyword (without ALL) combines these answer sets and eliminates duplicate rows from the combined answer set. UNION ALL combines the results of two or more queries without eliminating the duplicates. Column headings for the result are taken from the first query. The ORDER BY clause, if any, must be written after the last query in the UNION and is applied to the combined result that is displayed.

The UNION operator is useful when you want to merge lists of values based on more than one search condition, and when you want to indicate in the result which condition or conditions caused the resulting row to be selected.

The following query lists all projects for which the estimated mean number of employees is greater than .50, and the proportion of employee time spent on the project is greater than .5:

```
select projno, 'mean' -
from proj_act -
where acstaff > .50 -
union -
select projno, 'proportion' -
from emp_act -
where emptime > .5
```

In Figure 91, generated by the previous union query, the constants `mean` and `proportion` identify which query produced a particular row.

```
PROJNO  EXPRESSION 1
------  --------------
AD3110  MEAN
AD3110  PROPORTION
AD3111  MEAN
AD3111  PROPORTION
AD3112  MEAN
AD3112  PROPORTION
AD3113  MEAN
AD3113  PROPORTION
IF1000  MEAN
IF1000  PROPORTION
IF2000  MEAN
IF2000  PROPORTION
MA2100  MEAN
MA2100  PROPORTION
MA2110  MEAN
MA2110  PROPORTION
MA2111  MEAN
MA2111  PROPORTION
MA2112  MEAN
MA2112  PROPORTION
MA2113  MEAN
MA2113  PROPORTION
```

Figure 91. A Query Result from Using UNION

The query result appears to be ordered by PROJNO even though the example did not contain an ORDER BY clause. The database manager determines the order of the rows when it processes a UNION; therefore, the order of the rows on your display may differ from the order shown in Figure 91.

The database manager first executes the query on the PROJ_ACT table and forms an answer. It then performs the same activities on the EMP_ACT table. Then it combines the answers from the queries, sorts the rows, and eliminates duplicate rows.

The data types of the corresponding items in the SELECT clauses of all of the queries being merged must be compatible but need not be identical.

For a collection of queries to be connected by the UNION operator, they must satisfy the following rules:

- The data types of corresponding items in the SELECT clauses of all the queries must be compatible. For example, if the first column in a SELECT clause has a column type of CHAR, the first column of the SELECT clause of *each* query must be CHAR or VARCHAR; if the first column in a SELECT clause has a column type of INTEGER, the first item of the SELECT clause of *each* query must be any numeric type. Refer to the *DB2 Server for VSE & VM SQL Reference* manual for combinations of data types that can be used in the UNION operation.
- Corresponding items in the SELECT clauses need not have the same name. For example, a query beginning SELECT X can be united with a query that begins SELECT Y, if the X and Y have compatible data types.
- Each of the queries connected by a UNION or UNION ALL must have the same number of columns.
- If the results of the union are to be ordered, the ORDER BY clause must be written after the last query in the union. The order list must also indicate the *position* of an item in the select list, not the column name. To illustrate, *ORDER BY 1* is acceptable; *ORDER BY PROJNO* is not.
- A UNION cannot occur inside a subquery.
- A UNION cannot be used in the definition of a view.

If a sort is needed because of the union operation, consideration must be given to the length of the encoded key that is built by the sort. The length of the encoded key must not exceed 255 bytes, and is the sum of the lengths of the columns plus 25% of the lengths of any columns that are of varying-length character type.

Comparing the UNION and UNION ALL Operators

Use UNION ALL if no duplicate rows exist or if duplicate rows can be tolerated. UNION ALL results in better performance because no sorting is done to eliminate duplicates.

When you use both *UNION* and *UNION ALL* in a single statement, the result of the operation depends on the order of evaluation. Subqueries are evaluated from left to right. The following example contains no duplicates, because the result of the third query (UNION Q3) removes all duplicates:

```
Q1 UNION ALL Q2 UNION Q3
```

The following example contains no duplicates if Q2 contains rows that are duplicates of rows resulting from the union between Q3 and Q1:

```
Q3 UNION Q1 UNION ALL Q2
```

Using the UNION ALL Operator

In contrast to a UNION operator, a UNION ALL operator displays the results of two or more queries without eliminating any duplicates and without sorting the rows.

For example, type the previous SELECT statement with UNION ALL substituted for the UNION operator:

```
select projno,'mean' -
from proj_act -
where acstaff > .50 -
union all -
select projno,'proportion' -
from emp_act -
where emptime > .5
```

This displays the result shown in Figure 92.

```
PROJNO  EXPRESSION 1
------  --------------
AD3110  PROPORTION
AD3111  PROPORTION
AD3111  PROPORTION
AD3111  PROPORTION
AD3111  PROPORTION
AD3112  PROPORTION
AD3112  PROPORTION
AD3112  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
AD3113  PROPORTION
IF1000  PROPORTION
IF2000  PROPORTION
MA2100  PROPORTION
MA2110  PROPORTION
MA2111  PROPORTION
MA2111  PROPORTION
MA2111  PROPORTION
```

Figure 92. A Query Result from Using UNION ALL

The rows in this display are not sorted as they are for the UNION display. Using an ORDER BY clause displays sorted results for the UNION ALL query.

Using Parentheses in UNION Statements

You can also use parentheses to group subqueries in a UNION and UNION ALL statement to affect the result of the operation. When parentheses are included, the subqueries in the parentheses are evaluated first, followed by, from left to right, the other components in the statement.

As a result, the following operation:

```
(Q1 UNION ALL Q2) UNION Q3
```

can be quite different from:

```
(Q3 UNION Q1) UNION ALL Q2
```

Using Subqueries in Unions

You can also use subqueries in UNIONs. Before you begin the following example, set your terminal to process mixed case characters by typing:

```
set case string
```

The new setting permits the database manager to compare the mixed case characters in your query (in single quotation marks) with the mixed case characters in the ACTDESC column of the ACTIVITY table. The query:

```
select acstaff,acstdate,'Doc start' -
from proj_act -
where actno = (select actno -
            from activity -
            where actdesc = 'Document') -
union -
select acstaff,acstdate,'Spec start' -
from proj_act -
where actno = (select actno -
            from activity -
            where actdesc = 'Write specs')
```

lists the mean number of employees and estimated start date for activities DOCUMENT and WRITE SPECS. The query result is shown in Figure 93.

```
ACSTAFF  ACSTDATE    EXPRESSION 1
-------  ----------  --------------
   0.50  1982-01-01  Spec start
   0.50  1982-06-01  Doc start
   0.50  1982-08-15  Doc start
   0.50  1982-10-01  Doc start
   0.75  1982-01-01  Spec start
   0.75  1982-03-01  Doc start
   1.00  1982-01-01  Spec start
   1.00  1982-04-15  Doc start
   1.00  1982-07-01  Doc start
   1.00  1982-07-15  Doc start
   1.00  1982-10-15  Doc start
* End of Result *** 11 Rows Displayed ***Cost Estimate is 1********************
```

Figure 93. A Query Result from Using Subqueries in Unions

If you want to use a character constant in the SELECT clause of only one of the queries in a union operation, the constant must correspond to items in the SELECT clauses of the other query or queries that have a data type of CHAR or VARCHAR. You type the character constant enclosed in quotation marks. For example, the EMPLOYEE table has the column:

Column Name	Data Type
lastname	varchar(15) not null

To enter a constant in a union operation involving this table, type the following query:

```
select lastname -
from employee -
union -
select 'thompson' -
from employee
```

The character constant `thompson` is enclosed in quotation marks.

Note: You cannot use a UNION inside a subquery.

Reset your terminal to process uppercase characters by typing:

```
set case upper
```

Using Additional View Techniques

Views are described in "Using Views to Simplify Queries" on page 148 and "Naming Columns in a View" on page 149. These sections show you how to create views and name view columns that contain calculated values. Views are again mentioned in "Using a View to Restrict Privileges to Certain Rows" on page 190 and "Using a View to Restrict Privileges to Certain Columns" on page 190. Here you use them to manage your tables by limiting other users' access to your tables. Simplifying queries and preventing access to data are the two major reasons views are created.

For further practice, create the view PROJ1 as follows:

```
create view proj1 (proj_num,activ_num,mean_num) -
as select projno,actno,acstaff -
   from proj_act -
   where projno in (select projno -
               from project -
               where projname like '%program%')
```

The PROJ1 view contains the information shown in Figure 94 on page 157. (Recall that views are only imaginary tables that contain the results of a query.)

To see the contents of the PROJ1 view, type:

```
select * -
from proj1
```

```
PROJ_NUM  ACTIV_NUM  MEAN_NUM
--------  ---------  --------
AD3111           60      0.80
AD3111           60      0.50
AD3111           70      1.50
AD3111           70      0.50
AD3111           80      1.25
AD3111           80      1.00
AD3111          180      1.00
AD3112           60      0.75
AD3112           60      0.50
AD3112           60      0.75
AD3112           60      1.00
AD3112           70      0.75
AD3112           70      0.50
AD3112           70      1.00
AD3112           70      0.25
AD3112           80      0.35
AD3112           80      0.50
AD3112          180      0.50
AD3113           60      0.75
AD3113           60      1.00
```

Figure 94. Contents of the PROJ1 View

This view renamed and rearranged columns, and omitted unwanted columns and rows. You could have also defined the view's columns by calculating expressions, grouping results, or combining more than one table. Any SELECT statement that does not contain an ORDER BY clause or UNION can be used as the base of a view. The selected columns and rows become the view's columns and rows.

You can now use the view PROJ1 in the same manner as you use a table in queries, subqueries, or unions. The SQL statement that you type to query a view is combined with the definition of the view by the database manager to form an SQL statement that queries the underlying table.

Joining Views with Tables

A query that joins a view and a table is written just like a query that joins tables.

To retrieve the name and number of the project, as well as the activity number and description for all of the items in PROJ1, type the following join query:

```
select projname,proj_num,activ_num,actdesc -
from project,proj1,activity -
where projno = proj_num and activ_num = actno -
order by 2,3
```

This displays the result shown in Figure 95 on page 158.

```
PROJNAME                  PROJ_NUM  ACTIV_NUM  ACTDESC
------------------------  --------  ---------  --------------------
PAYROLL PROGRAMMING       AD3111           60  Describe logic
PAYROLL PROGRAMMING       AD3111           60  Describe logic
PAYROLL PROGRAMMING       AD3111           70  Code programs
PAYROLL PROGRAMMING       AD3111           70  Code programs
PAYROLL PROGRAMMING       AD3111           80  Test programs
PAYROLL PROGRAMMING       AD3111           80  Test programs
PAYROLL PROGRAMMING       AD3111          180  Document
PERSONNEL PROGRAMMG       AD3112           60  Describe logic
PERSONNEL PROGRAMMG       AD3112           60  Describe logic
PERSONNEL PROGRAMMG       AD3112           60  Describe logic
PERSONNEL PROGRAMMG       AD3112           60  Describe logic
PERSONNEL PROGRAMMG       AD3112           70  Code programs
PERSONNEL PROGRAMMG       AD3112           70  Code programs
PERSONNEL PROGRAMMG       AD3112           70  Code programs
PERSONNEL PROGRAMMG       AD3112           70  Code programs
PERSONNEL PROGRAMMG       AD3112           80  Test programs
PERSONNEL PROGRAMMG       AD3112           80  Test programs
PERSONNEL PROGRAMMG       AD3112          180  Document
ACCOUNT PROGRAMMING       AD3113           60  Describe logic
ACCOUNT PROGRAMMING       AD3113           60  Describe logic
```

Figure 95. A Query Result from Joining a View with a Table

Joining Views with Other Views

Besides joining a view with a table, you can join a view with another view. Create the view PROJ2 as follows:

```
create view proj2 (proj_nmbr,activ_nmbr,mean_num) -
as select projno,actno,acstaff -
   from proj_act -
   where projno = 'AD3111'
```

PROJ2 contains the information shown in Figure 96. Query PROJ2 to produce the display.

```
PROJ_NMBR  ACTIV_NMBR  MEAN_NUM
---------  ----------  --------
AD3111             60      0.80
AD3111             60      0.50
AD3111             70      1.50
AD3111             70      0.50
AD3111             80      1.25
AD3111             80      1.00
AD3111            180      1.00
* End of Result ************** 7 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 96. Content of PROJ2 View

You can now use PROJ1 and PROJ2 in the same query or queries. Type:

```
select proj_num,activ_num,p1.mean_num -
from proj1 p1,proj2 p2 -
where proj_num = proj_nmbr -
and activ_num = activ_nmbr -
and p1.mean_num = p2.mean_num
```

and you see the result in Figure 97 on page 159.

```
PROJ_NUM  ACTIV_NUM  MEAN_NUM
--------  ---------  --------
AD3111           60      0.80
AD3111           60      0.50
AD3111           70      1.50
AD3111           70      0.50
AD3111           80      1.25
AD3111           80      1.00
AD3111          180      1.00
* End of Result ************** 7 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 97. A Query Result from Joining Two Views

This query found the information in PROJ1 that is also contained in PROJ2. You can also produce this information by testing for existence, which is discussed in the next section.

Testing for Existence

You test for existence to determine if a row exists that satisfies a certain condition. Type the following query:

```
select proj_num,activ_num,mean_num -
from proj1 -
where exists (select * -
          from proj2 -
          where proj1.proj_num = proj_nmbr -
          and proj1.activ_num = activ_nmbr)
```

The subquery is processed until one row is found that satisfies that subquery, or until every row is checked and none found that satisfies the subquery. In the SELECT clause of the subquery, the * returns all columns of the table for each row satisfying the WHERE condition. Because you only want to know if the subquery would return data, you need not select information from a particular table column.

This query produces a result identical to the query in the previous section as you can see by comparing Figure 97 and Figure 98.

```
PROJ_NUM  ACTIV_NUM  MEAN_NUM
--------  ---------  --------
AD3111           60      0.80
AD3111           60      0.50
AD3111           70      1.50
AD3111           70      0.50
AD3111           80      1.25
AD3111           80      1.00
AD3111          180      1.00
* End of Result ************** 7 Rows Displayed ****** Cost Estimate is  1   **
```

Figure 98. A Query Result Showing the Existence of Rows That Meet Conditions

The syntax for the EXISTS search condition in a query is:

```
►►──┬─────┬──EXISTS──(subquery)──────────────────────────►◄
    └─NOT─┘
```

The WHERE clause is satisfied if any rows are returned by the subquery. Use NOT EXISTS if you want the WHERE clause to be satisfied if no rows are returned by the subquery.

To find the information for projects in PROJ1 that are not in PROJ2, type:

```
select projname,proj_num,actdesc -
from project,proj1,activity -
where not exists (select proj_nmbr -
                  from proj2 -
                  where proj_nmbr = proj1.proj_num) -
and activ_num = actno -
and proj_num = projno
```

The results are shown in Figure 99.

Note that AND connects the other search conditions to the NOT EXISTS function.

```
PROJNAME              PROJ_NUM  ACTDESC
--------------------- --------  --------------------
PERSONNEL PROGRAMMI    AD3112    Code programs
PERSONNEL PROGRAMMI    AD3112    Code programs
PERSONNEL PROGRAMMI    AD3112    Code programs
PERSONNEL PROGRAMMI    AD3112    Code programs
ACCOUNT PROGRAMMING    AD3113    Code programs
ACCOUNT PROGRAMMING    AD3113    Code programs
ACCOUNT PROGRAMMING    AD3113    Code programs
ACCOUNT PROGRAMMING    AD3113    Code programs
ACCOUNT PROGRAMMING    AD3113    Code programs
PERSONNEL PROGRAMMI    AD3112    Document
ACCOUNT PROGRAMMING    AD3113    Document
ACCOUNT PROGRAMMING    AD3113    Document
ACCOUNT PROGRAMMING    AD3113    Document
W L PROGRAM DESIGN     MA2111    Lead program/design
PERSONNEL PROGRAMMI    AD3112    Describe logic
PERSONNEL PROGRAMMI    AD3112    Describe logic
PERSONNEL PROGRAMMI    AD3112    Describe logic
PERSONNEL PROGRAMMI    AD3112    Describe logic
PERSONNEL PROGRAMMI    AD3112    Describe logic
ACCOUNT PROGRAMMING    AD3113    Describe logic
ACCOUNT PROGRAMMING    AD3113    Describe logic
ACCOUNT PROGRAMMING    AD3113    Describe logic
```

Figure 99. A Query Result from Using NOT EXISTS

Using Views in Unions

In addition to tables, you can also merge views using the UNION operator. You can use tables and views or only views when you merge the information. Type:

```
select * -
from proj1 -
where mean_num >= 1.00 -
union -
select * -
from proj2 -
where mean_num >= 1.00
```

The result of this query is in Figure 100 on page 161.

```
PROJ_NUM  ACTIV_NUM  MEAN_NUM
--------  ---------  --------
AD3111           70      1.50
AD3111           80      1.00
AD3111           80      1.25
AD3111          180      1.00
AD3112           60      1.00
AD3112           70      1.00
AD3113           60      1.00
AD3113           70      1.00
AD3113           70      1.25
AD3113           80      1.75
AD3113          180      1.00
MA2110           10      1.00
MA2111           40      1.00
MA2111           50      1.00
MA2111           60      1.00
* End of Result *** 15 Rows Displayed ***Cost Estimate is 1********************
```

Figure 100. A Query Result from Using Views in Unions

Notice that the column names of the *first* SELECT statement name the columns in the query result. Remember, however, that corresponding columns must have the same data type.

If you use an ORDER BY clause in a UNION operation, you *cannot* use the column name after ORDER BY. Use a column number, as shown in the following query.

```
select prstdate,prendate,'proj1' -
from project -
where projno in (select proj_num -
             from proj1) -
union -
select prstdate,prendate,'proj2' -
from project -
where projno in (select proj_nmbr -
             from proj2) -
order by 2,3
```

The columns are ordered primarily by the estimated project end date and secondarily by the constant.

The result of these merged queries is in Figure 101.

```
PRSTDATE    PRENDATE    EXPRESSION 1
----------  ----------  --------------
1982-01-01  1982-12-01  PROJ1
1982-01-01  1983-02-01  PROJ1
1982-01-01  1983-02-01  PROJ2
* End of Result *** 3 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 101. A Query Result from Using Views in Unions with the ORDER BY Clause

Updating Tables on Which Views are Defined

Like SELECT statements, the INSERT, DELETE, and UPDATE statements can be applied to a view as though it is an actual, stored table. When you change a table by using an update statement on either a table or a view, this change can be seen by the users of the view, of the underlying table, or of other views defined on the same data. The SQL statement that operates on the view is combined with the view definition to form an SQL statement that operates on the related table.

To illustrate updating, consider the view PROAC, which is created using the following statement:

```
create view proac (proj,activ,est_mean,est_start) -
as select projno,actno,acstaff,acstdate -
   from proj_act -
   where acstaff > 1.00
```

You can update this view using a statement such as:

```
update proac -
set est_start = '1982-07-15' -
where proj = 'AD3113' -
and activ = 70
```

To apply this change, the database manager updates the underlying table, PROJ_ACT, with an update operation equivalent to:

```
update proj_act -
set acstdate = '1982-07-15' -
where projno = 'AD3113' -
and actno = 70
```

The following limitations must be observed when modifying a table through a view:

1. INSERT, DELETE, and UPDATE of the view are not permitted if the view involves any of the following operations: join, GROUP BY, HAVING, DISTINCT, or any column function such as AVG. If one or more of these operations is present in the view definition, the owner of the view does not receive INSERT, DELETE, or UPDATE privileges on the view.
2. A column of a view can be updated only if it is derived directly from a column of a stored table. Columns defined by expressions such as *EMPTIME+.25* cannot be updated. If a view is defined containing one or more such columns, the owner does not receive update privileges on these columns. INSERT statements are not permitted on views containing such columns; DELETE statements are permitted.
3. ALTER TABLE, CREATE INDEX, and UPDATE STATISTICS statements cannot be applied to a view.
4. INSERT, DELETE, and UPDATE statements, as well as INPUT commands, cannot be applied to a view defined on a read-only view.

An INSERT statement can be used with a view that does not contain all the columns of the stored underlying table. For example, it is possible to insert rows into the view PROAC even though it does not contain the column ACENDATE of the underlying table PROJ_ACT. When such an insert is performed, the underlying table column is given a null value.

Note: Insertions to a view are not permitted in cases where any underlying table column that does not permit null values is omitted from the view.

If a view is created without the WITH CHECK OPTION, you can perform insert or update operations on that view, even though the insert or update operation does not satisfy the view definition. For example, the view PROAC is defined by the condition ACSTAFF>1.00. Inserting rows into PROAC that contain a value less than 1.00 in the EST_MEAN field (the field defined as ACSTAFF), or updating a row of PROAC so that its EST_MEAN value becomes less than 1.00, is possible because the insert or update operations are performed on the underlying PROJ_ACT table.

If you want to make sure that the insert or update operation conforms with the view definition, use the WITH CHECK OPTION clause as described in "Checking Inserts or Updates in a View" on page 165.

Be careful when updating tables through views that may contain duplicate rows. For example, suppose a view ACTIV, which is defined on the ACTIVITY table, contains only the columns ACTKWD and ACTDESC from ACTIVITY. Because the primary-key column ACTNO, whose presence ensures unique rows, is not included in the view, duplicate row entries can exist for the ACTKWD and ACTDESC columns. When you update the ACTIVITY table using the ACTIV view, all the duplicate rows, if they exist, are also updated.

Another area for caution involves inserting information into a table and then querying the table through its associated view. Depending on how the view is defined, the new table information may or may not be displayed in the view query. To illustrate, consider the PROJ1 view you created for the PROJ_ACT table in "Using Additional View Techniques" on page 156. Start by making the following two insertions into the PROJ_ACT table:

```
insert into proj_act -
values ('MA2110',50,.50,'1982-01-01','1982-06-01')

insert into proj_act -
values ('PL2100',50,.50,'1982-01-01','1982-06-01')
```

Now when you query the table by querying the PROJ1 view, you see the display in Figure 102 on page 164.

```
PROJ_NUM  ACTIV_NUM  MEAN_NUM
--------  ---------  --------
AD3111           60      0.80
AD3111           60      0.50
AD3111           70      1.50
AD3111           70      0.50
AD3111           80      1.25
AD3111           80      1.00
AD3111          180      1.00
AD3112           60      0.75
AD3112           60      0.50
AD3112           60      0.75
AD3112           60      1.00
AD3112           70      0.75
AD3112           70      0.50
AD3112           70      1.00
AD3112           70      0.25
AD3112           80      0.35
AD3112           80      0.50
AD3112          180      0.50
AD3113           60      0.75
AD3113           60      1.00
AD3113           60      0.25
AD3113           70      0.50
```

Figure 102. A Query Result Showing What New Table Data Is Displayed in a View

If you scroll down, you see that the new row for project MA2110, activity 50, appears, but the new row for project PL2100, activity 50, does not. Project PL2100 is not selected because its project name does not meet the view requirements that it contain the character string *program*.

You can also change the underlying table by performing update commands on the view. But again, exercise caution. The following example illustrates why:

```
insert into proj1 -
values ('IF1000',50,.75)
```

This insert operation fails because it attempts to insert null values into the PROJ_ACT columns ACSTDATE and ACENDATE, and ACSTDATE requires values.

Creating a View Defined on Another View

You create a view defined on another view the same way you create a view defined on a table—by using the CREATE VIEW statement.

To create view PROJ3, which contains the activity numbers and estimated mean number of employees from the view PROJ2, which you created earlier, type:

```
create view proj3 (act, mean) -
as select activ_nmbr,mean_num -
   from proj2
```

PROJ3 contains the information shown in Figure 103 on page 165. Query PROJ3 to display these results.

```
   ACT     MEAN
------  -------
    60     0.80
    60     0.50
    70     1.50
    70     0.50
    80     1.25
    80     1.00
   180     1.00
* End of Result *** 7 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 103. Contents of a View Defined on Another View

You can use PROJ3 in the same kinds of queries and unions as you used PROJ1 or PROJ2.

Note: SELECT is the only operation you can use, or grant as a privilege, on a view defined on another view.

Checking Inserts or Updates in a View

To ensure all insert and update operations are done according to your view definition, you can include the WITH CHECK OPTION clause in your CREATE VIEW statement.

This clause lets you check all inserts and updates on a view against the view definition that you specify. If the associated value or values do not fall within the definition you created, the inserts or updates are not allowed.

Consider the following example:

```
create view v1 -
as select * from employee -
   where salary <= 50000
```

You can update SALARY to a value higher than 50000 even though such a row cannot be selected.

However, if the WITH CHECK OPTION clause is added, as in the following definition, you can update SALARY to a value up to and including 50000 only:

```
create view v1 -
as select * from employee -
   where salary <= 50000 -
   with check option
```

You cannot use the WITH CHECK OPTION clause on views that cannot be updated or views that are built on subqueries.

If a view depends on other views, checking is performed according to the following rules:

- If a view and the views on which it depends use the WITH CHECK OPTION clause, all inserts and updates to this view are checked against the definitions of the view and the views on which it depends.
- If this view uses the WITH CHECK OPTION clause, but none of the views on which it depends do so, all inserts and updates to this view will be checked against the definition of this view only.

- If this view does not use the WITH CHECK OPTION clause, but some views on which it depends use it, all inserts and updates to this view will be checked only against the definitions of the views that use the WITH CHECK OPTION clause.
- If no view uses the WITH CHECK OPTION clause, no checking is performed.

Listing Information about the Views You Create

You have created several views. To display their names, you can use the query on page 175 to access the catalog table SYSCATALOG. But a problem arises here in that the query does not indicate whether the names listed refer to tables or views. To display only the names of the views you have created, type:

```
select tname -
from system.syscatalog -
where creator = user and tabletype = 'v'
```

This query displays a result similar to Figure 104.

```
TNAME
------------------
SYSUSERLIST
ACT10
ACT60
EMPLS
PROJ1
PROJ2
PROAC
PROJ3
* End of Result *** 8 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 104. A Query Result That Displays the Names of the Views You Created

Use a TABLETYPE of *R* to produce a list of table names.

You can also query the SYSVIEWS system catalog. It indicates the name of the view, the SQL statement that defined or created the view, and other information. To determine the view names and definition statement, type:

```
select viewname,viewtext -
from system.sysviews -
where vcreator = user
```

This query displays a result similar to Figure 105 on page 167.

```
VIEWNAME            VIEWTEXT
------------------  --------------------
ACT10               CREATE VIEW ACT10 AS<
ACT60               CREATE VIEW ACT60 (P<
EMPLS               CREATE VIEW EMPLS (A<
PROAC               CREATE VIEW PROAC (P<
PROJ1               CREATE VIEW PROJ1 (P<
PROJ2               CREATE VIEW PROJ2 (P<
PROJ3               CREATE VIEW PROJ3 (A<
SYSUSERLIST         CREATE VIEW SQLDBA.S<
* End of Result *** 8 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 105. A Query Result Showing the Names and Creations of Views

The column VIEWNAME contains the name of the view. The VIEWTEXT column contains the first 20 characters of the view definition. The character < at the end of each line under this column indicates that the view definition statement is longer than 20 characters. Only 20 characters are shown because that is the value to which VARCHAR is set in the profile routine, but this is not always the case (see “Selecting Particular Columns” on page 21). You can change this setting by using the SET VARCHAR command described in “SET” on page 245.

Dropping Views

You can drop (delete) your views from the database using the DROP VIEW statement. Enter the following to drop PROJ2:

```
drop view proj2
```

The database manager automatically deletes all other views that refer to the deleted view; PROJ3 is also deleted. If you now try to use PROJ3 in a query, you receive the following error message:

```
ARI0503E An SQL error has occurred.
         SQLDBA.PROJ3 was not found in the system catalogs.
ARI0505I SQLCODE = -204 SQLSTATE = 52004 ROWCOUNT = 0
ARI0504I SQLERRP: ARIXOCA SQLERRD1: -100 SQLERRD2: 0
```

The underlying table(s) on which the view(s) are defined are not affected by DROP VIEW statements.

Computing Percentages

Computing percentages is not a function of the database manager. You can, however, compute percentages by using the following techniques.

In general, two passes through a table are required to compute percentages. An auxiliary table holds the result of the first pass.

1. The first pass develops the totals of interest and places them in the auxiliary table. Use a Format 2 INSERT statement that contains a SELECT statement.
2. The second pass uses the result of the first pass to compute the percentages. Use a SELECT statement to join the first table to the second table. Finally, ISQL FORMAT commands can be used to prepare the final display or report.

The following examples illustrate the calculation of:

- A simple percentage — a detail row divided by the total.
- Aggregate percentages — a grouped row value divided by a total for the group or by the overall total.

  Note: In some of the examples, the percentage values may not total 100% because of truncation.

Example of Computing a Simple Percentage

This example uses a table defined as:

```
create table sales (item   char(8) not null, -
                    quantity  integer, -
                    amount  decimal(9,2))
```

You can input the table data that is shown below. See "Inserting Table Data" on page 81 if you want to know how to input the data.

```
Sales Table:

       ITEM     QUANTITY     AMOUNT

       1111       100       5500.50
       2222       200      30000.50
       3333       300       4780.10
```

To calculate percentages, the totals must be computed first. The totals are held in an auxiliary table defined as:

```
create table total (quantity  integer, -
                    amount  decimal(9,2))
```

The totals are calculated and stored in the auxiliary table by this INSERT command:

```
insert into total -
   select sum(quantity),sum(amount) -
   from sales
```

By joining each row in the SALES table with the row in the TOTAL table, you can calculate and display the percentages with this SELECT statement:

```
select item, -
     sales.quantity, -
     sales.quantity * 100/total.quantity, -
     sales.amount, -
     sales.amount * 100/total.amount -
  from sales,total
```

The addition of these FORMAT commands gives a result similar to that shown in Figure 106 on page 169:

```
format column 3 name '% OF TOTAL QTY.'
format column 5 name '% OF TOTAL AMT.'
format total (2 3 4 5)
```

```
Result Display:


ITEM     QUANTITY     % OF TOTAL QTY.     AMOUNT     % OF TOTAL AMT.
----     --------     ---------------     --------   ---------------
1111       100            16.66            5500.50        13.65
2222       200            33.33           30000.50        74.47
3333       300            50.00            4780.10        11.86
====     ========     ==============     =========   ===============
           600            99.99           40281.10        99.98
```

Figure 106. Simple Percentage Example

Example of Computing an Aggregate Percentage

This example uses a table defined as:

```
create table regionsales (region    char(3), -
                          sales     char(3), -
                          item      char(8), -
                          quantity  integer, -
                          amount    decimal(9,2))
```

This table holds the sales records of the items by region. You can input the table data that is shown below. See “Inserting Table Data” on page 81 if you want to know how to input the data.

Regionsales Table:

REGION	SALES	ITEM	QUANTITY	AMOUNT
001	AAA	1111	60	3300.30
001	BBB	1111	40	2200.20
001	AAA	2222	100	15000.25
001	BBB	2222	100	15000.25
001	CCC	3333	300	4780.10
002	DDD	1111	90	4950.45
002	EEE	2222	20	3000.05
002	DDD	3333	20	318.67
002	EEE	3333	130	2071.38

To calculate aggregate percentages, you must compute the totals first. The totals are held in an auxiliary table defined as:

```
create table totals (id       char(1), -
                     region   char(3), -
                     quantity integer, -
                     amount   decimal(9,2))
```

The grand totals are calculated by this INSERT command:

```
insert into totals (id,quantity,amount) -
   select 't',sum(quantity),sum(amount) -
   from regionsales
```

The regional totals are calculated by this INSERT command:

```
insert into totals (id,region,quantity,amount) -
   select 'r',region,sum(quantity),sum(amount) -
   from regionsales -
   group by region
```

By joining each row in the REGIONSALES table with the T row in the TOTALS table grouped by item, you can calculate and display the aggregate percentage for each item to the grand totals with this SELECT statement:

```
select item, -
 sum(regionsales.quantity), -
 sum(regionsales.quantity) * 100 -
   /(sum(totals.quantity)/count(*)), -
 sum(regionsales.amount), -
 sum(regionsales.amount) * 100 / (sum(totals.amount)/count(*)) -
from regionsales,totals -
where id = 't' -
group by item -
order by item
```

The addition of these FORMAT commands gives a result similar to that shown in Figure 107:

```
format column 3 name '% OF TOTAL QTY.'
format column 5 name '% OF TOTAL AMT.'
format total (2 3 4 5)
```

```
Result:(Aggregate percentages of each item to the grand totals)



ITEM  SUM(QUANTITY)  % OF TOTAL QTY.   SUM(AMOUNT)  % OF TOTAL AMT.
----  -------------  ---------------  ------------  ---------------
1111            190               22      10450.95              20.
2222            220               25      33000.55              65.
3333            450               52       7170.15              14.
      =============  ===============  ============  ===============
                860               99      50621.65              99.
```

Figure 107. Aggregate Percentage Example — by Item

By joining each row in the REGIONSALES table with the R and T rows in the TOTALS table grouped by region and salesperson, you can calculate and display the aggregate percentage for each salesperson by region and the grand total with this SELECT statement:

```
select regionsales.region,sales, -
  sum(regionsales.quantity), -
  sum(regionsales.quantity) * 100 / (sum(r.quantity)/count(*)), -
  sum(regionsales.quantity) * 100 / (sum(t.quantity)/count(*)), -
  sum(regionsales.amount), -
  sum(regionsales.amount) * 100 / (sum(r.amount)/count(*)), -
  sum(regionsales.amount) * 100 / (sum(t.amount)/count(*)) -
from regionsales,totals r,totals t -
where r.id = 'r' and -
  t.id = 't' and -
  r.region = regionsales.region -
group by regionsales.region,sales -
order by regionsales.region,sales
```

Note: `count(*)` represents the number of occurrences of the item in the group; therefore, it is the number of joins with the T row in the TOTALS table.

The addition of these FORMAT commands gives a result similar to that shown in Figure 108:

```
format column 4 name '% REGION'
format column 5 name '% TOTAL'
format column 7 name '% REGION'
format column 8 name '% TOTAL'
format group 1
format subtotal (3 4 5 6 7 8)
format total except (4 7)
```

```
REGION  SALES  SUM(QUANTITY)  % REGION  % TOTAL  SUM(AMOUNT)  % REGION  % TOTAL
------  -----  -------------  --------  -------  -----------  --------  -------
   001    AAA            160        26       18     18300.55       45.      36.
          BBB            140        23       16     17200.45       42.      33.
          CCC            300        50       34      4780.10       11.       9.
               -------------  --------  -------  -----------  --------  -------
   ***                   600        99       68     40281.10       98.      78.

   002    DDD            110        42       12      5269.12       50.      10.
          EEE            150        57       17      5071.43       49.      10.
               -------------  --------  -------  -----------  --------  -------
   ***                   260        99       29     10340.55       99.      20.
               =============            =======  ===========            =======
                         860                 97     50621.65                98.
```

Figure 108. Aggregate Percentage Example — by Region

Cancelling Running Commands

It may take extended time for your commands, statements, or routines to be performed. The processing of your query statement, for example, can be delayed if another user is accessing the same table.

DB2 Server for VM

You can query the status of the command, statement or routine by issuing the CMS Immediate command, SQLQRY. The SQLQRY command returns information that can help you to determine the cause of the delay. For more information on the SQLQRY command, refer to the *DB2 Server for VM Database Administration* manual.

You have the option of cancelling in-progress commands (such as the ISQL INPUT command), long-running SQL commands (in which case message ARI7044I is issued in VSE only), or LUWs (where AUTOCOMMIT is off), by issuing a CANCEL command. You can type this command anytime during the typing of commands, statements, or data.

To cancel an operation, type:

```
cancel
```

DB2 Server for VSE

This command also performs a ROLLBACK operation.

To cancel a long-running command, wait until you receive the message:

```
ARI7044I  Command in progress. Terminal is now free.
```

Press CLEAR, and type:

```
isql cancel
```

If the ISQL user is connected to a remote DRDA application server, the user cannot cancel long-running SQL commands. In this case, message ARI7044I is not displayed.

DB2 Server for VM

This command also causes a ROLLBACK RELEASE operation to be performed. The database manager releases your connection to the application server that you are using. If you previously issued an explicit CONNECT command to connect to a particular application server, you must reissue another CONNECT command. If you do not, the next statement or command you type causes you to be implicitly connected to the default application server. If you have been implicitly connected to a default application server, your next statement or command implicitly reconnects you.

When you cancel a routine that is in progress, the routine stops execution immediately, and you are left in ISQL.

For information on the effects of AUTOCOMMIT on the CANCEL command, see "Using the AUTOCOMMIT ON Setting" on page 71, or the description of the CANCEL command in Chapter 13, "ISQL Commands" on page 203.

EXERCISE 15

(Answers are in Appendix A, Answers to the Exercises, on page 267.)

Perform the following:

1. Select the project number, project name, project staff, activity number, and activity staff for those activities where the total activity staff adds up to more than half the project staff for that project. Order the results by project number.
2. Retrieve the project number, activity number, activity staff, employee time, and employee number, where the employee time is less than half the average activity staff for that project, and there are at least three activity numbers for that project.
3. Retrieve the total commission for the MANUFACTURING SYSTEMS and the ADMINISTRATION SYSTEMS departments. Include the constant *MANUFACTURING SYSTEMS* for department D11 and the constant *ADMINISTRATION SYSTEMS* for department D21.
4. Use the union operation to select the employee number, activity number, and employee time for project W L PROGRAMMING and project number MA2112.
5. Create the view MANAGERS on the EMPLOYEE table for all information where the job is manager.
6. Retrieve the last name, project number, and activity number for each row in MANAGERS.
7. Select the average, maximum, and minimum salary from MANAGERS and EMPLOYEE. Use the constants MANAGERS and EMPLOYEES to identify the results.
8. Drop the MANAGERS view.
9. Determine which views you have defined.
10. Drop the views you created in this chapter.
11. Delete the three rows you added to the PROJ_ACT table in this chapter.

Chapter 11. Creating and Managing Tables

Some users of the database manager are authorized to create and manage their own set of tables. There are two ways you can obtain the authority to create and manage your own tables. You may have RESOURCE authority, or you may ask someone with DBA authority to acquire a dbspace on your behalf. The latter lets you create tables in a private dbspace only (dbspaces are discussed later in this chapter.)

To determine what authority you have, type:

```
select resourceauth from sqldba.sysuserlist -
where name = user
```

A `Y` under `RESOURCEAUTH` indicates that you have resource authority and can create and manage your own set of tables.

You can use the following sections if you have resource authority. They describe how to manage your own tables, share them with other users, and improve query performance.

Managing Your Own Tables

Managing your tables consists of the following:

- Determining what your information is
- Creating tables
- Creating referential structures
- Determining where your tables are stored
- Dropping (deleting) tables
- Dropping primary or foreign keys
- Activating or deactivating primary or foreign keys
- Copying data from one table into another
- Identifying the minimum contents of a table
- Adding columns to a table.

Querying Information about Your Tables

The database manager automatically maintains a catalog that contains several catalog tables. The catalog holds information about the application server. Table names, view names, table owners, view owners, and column names are just a part of the information that can be found in the catalog tables. The data in the catalog tables is available to SQL users through the normal SQL query facilities. Whenever you use the catalog tables, remember to prefix the table name with the owner name *system*.

To find out which tables and views belong to you, type the following:

```
select tname,remarks -
from system.syscatalog -
where creator = user
```

This query presents results similar to Figure 109 on page 176.

```
TNAME               REMARKS
------------------  -----------------------------------
ACT10
ACT60
PROJ1
PROJ2
ROUTINE
* End of Result *** 5 Rows Displayed ***Cost Estimate is 1********************
```

Figure 109. A Query Result Displaying Which Tables and Views Belong to You

Your display may provide additional information if you have created views and tables of your own.

The column TNAME contains the table or view name. The REMARKS column contains information about the table or view. You enter information into this field for your tables or views using the COMMENT command. (For information on using the COMMENT command, see the *DB2 Server for VSE & VM SQL Reference* manual.)

USER in the above query instructs the system to use your authorization ID when selecting information from the SYSCATALOG table. You can use your authorization ID in place of USER in the WHERE clause. If your authorization ID is VELDA, for example, you can type the previous statement as:

```
select tname,remarks -
from system.syscatalog -
where creator = 'velda'
```

To determine the column names of the EMPLOYEE table, type the following:

```
select cname -
from system.syscolumns -
where tname = 'employee'
```

This query provides results similar to Figure 110.

```
CNAME
------------------
BIRTHDATE
BONUS
COMM
EDLEVEL
EMPNO
FIRSTNME
HIREDATE
JOB
LASTNAME
MIDINIT
PHONENO
SALARY
SEX
WORKDEPT
* End of Result *** 14 Rows Displayed ***Cost Estimate is 1********************
```

Figure 110. A Query Result Displaying the Column Names of the EMPLOYEE Table

For more information on the catalog, see the *DB2 Server for VSE & VM SQL Reference* manual.

Creating Your Own Tables

You can create your own table by providing the database manager with a name for the table and the columns that you want it to contain. The names chosen can consist of letters, numbers, and some special characters. If you want a blank or other special character in the name, you must enclose the name in double quotation marks.

When you provide the column names, you must also indicate the type of data for each column.

For example, type the following SQL statement to create a table with employee information for a particular quarter:

```
create table empl -
(empno char(6), -
 lastname varchar(15), -
 edlevel smallint, -
 birthdate date, -
 quarter integer)
```

Identifying Data Types

The DBCS data types support character sets that require 2 bytes of storage for each character in the character set. Kanji and APL are examples of such character sets.

DB2 Server for VM

If you are going to use double-byte characters during your ISQL session, you enter the CMS command:

```
set fullscreen on
```

before starting ISQL or from CMS subset mode (see “CMS-Subset Processing” on page 197.

Once you have created a table, you can insert data into it using the commands described in “Inserting Table Data” on page 81. The list below shows the types of data that can be defined for a column. See the *DB2 Server for VSE & VM SQL Reference* manual for more information.

INTeger
For large positive or negative whole numbers.

SMALLINT
For small positive or negative whole numbers. The largest number you can put in this field is 32767; the smallest, -32768.

DECimal or NUMERIC
DECimal (*p*) or NUMERIC (*p*)
DECimal (*p*,*s*) or NUMERIC (*p*,*s*)
For decimal data. The *p* identifies the total number of decimal digits a number can have. The maximum allowable precision is 31 digits. The *s* identifies the number of digits on the right side of the decimal point. The default value is DECimal(5,0). NUMERIC is a synonym for DECIMAL. If a column name is defined as NUMERIC, the column is treated like a DECIMAL data type.

FLOAT
FLOAT (*n*)
For floating-point numbers.

REAL
For single precision floating-point numbers.

DOUBLE PRECISION
For double precision floating-point numbers.

CHARacter
CHARacter(*n*)
For character data

that has a fixed number of characters (*n*). The maximum number of characters is 254.

VARCHAR(*n*)
For character data[1] that varies in length.

LONG VARCHAR
For character data[1] that varies in length up to 32767 characters.

GRAPHIC
GRAPHIC(*n*)
For double-byte character set (DBCS)

data that has a fixed number of DBCS characters (*n*). The maximum number of DBCS characters is 127.

VARGRAPHIC(*n*)
For a varying-length graphic string of maximum length *n*, which can range from 1 to 16383 double-byte characters.[2] If *n* is greater than 127 double-type characters, the string is defined as a long string column.

LONG VARGRAPHIC
For a varying-length string of double-byte characters, of maximum length 16383 bytes.[3] A LONG VARGRAPHIC column is always a long string column (even if its actual length is less than 128 double-byte characters).

VARGRAPHIC(*n*)
For a varying-length graphic string of maximum length *n*, which can range from 1 to 16383 double-byte characters. [2]

DATE
For a three-part value in several formats that designates a point in time according to the calendar.

TIME
For a three-part value in several formats that designates a time of day according to a 24-hour clock.

---

1 Character data types CHAR, VARCHAR, and LONG VARCHAR support the BIT, mixed, and SBCS data subtypes. See the *DB2 Server for VSE & VM SQL Reference* manual for more information on defining character data types.

2 ISQL does *not* support INSERT, UPDATE, or SELECT for tables or views with VARCHAR>254, VARGRAPHIC>127, LONG VARCHAR or LONG VARGRAPHIC columns.

3 ISQL supports hexadecimal constants and graphic constants that can be used to insert (INSERT or INPUT commands) or update data into DBCS columns of length <= 127. Hexadecimal constants and graphic constants can also be used in WHERE clauses with DBCS columns. See the *DB2 Server for VSE & VM SQL Reference* manual for more information about this data type.

TIMESTAMP
For a seven-part value that designates a date and time, including a fractional part. The seven parts are year, month, day, hour, minute, second, and microsecond.

Identifying Datetime Types

Five formats are available when you are typing datetime data. You can use any format for input. However, when the datetime data is displayed, or returned to the application program, the default format is used. For more information on Datetime data type formats, see the *DB2 Server for VSE & VM SQL Reference* manual.

Figure 111 summarizes the different formats you can use to type datetime values:

Figure 111. Date and Time Formats

Data Type	Data Type Description	Data Type Format
Date	ISO JIS EUR USA LOCAL	yyyy-mm-dd yyyy-mm-dd dd.mm.yyyy mm/dd/yyyy site defined
Time	ISO JIS EUR USA LOCAL	hh.mm[.ss] hh:mm[:ss] hh.mm[.ss] hh:mm xM site defined
Timestamp		yyyy-mm-dd-hh.mm.ss[.[zzzzzz]]

Where:

- ISO is the International Standards Organization form.
- JIS is the Japanese Industrial Standard Christian Era form.
- USA is the IBM standard for the United States form.
- EUR is the IBM standard for the European form.
- LOCAL is any site defined form.

and

- yyyy is the year.
- mm is the month.
- dd is the day.
- hh is the hour

  In USA format, the range of hh is 1–12 except for 00:00 am. In ISO, JIS, and EUR format, the range of hh is 0–24. For input purposes, the leading zeros can be suppressed.

- mm is the minute.
- ss is the second.
- zzzzzz is the microsecond.
- xM is AM or PM.
- [ ] indicates it is optional; default is zeros.

Note: LOCAL date and time is not supported on an application server that is accessed using the DRDA protocol.

A DBA can change the date and time default formats, which are defined in the SYSTEM.SYSOPTIONS table, from ISO (which is the system-supplied application server default form) to any site-defined format. The change becomes effective the next time the application server is started.

Date, time, and timestamp values are stored in their respective internal formats. They can be displayed to the user in one of the stated formats. You can use the CHAR scalar function to display the date or time value in the format you want. The datetime values can only be assigned to a column of data type CHAR, VARCHAR, or one of DATE, TIME, or TIMESTAMP.

Defining Column Data

When you create a table, you can indicate that character columns will contain single-byte characters, single or double-byte characters, or bit data by specifying FOR SBCS DATA, FOR MIXED DATA, or FOR BIT DATA respectively.

You can also indicate that character or graphic columns will contain data represented using a specific character set, code page, and encoding scheme by specifying a Coded Character Set Identifier (CCSID). CCSIDs are important for applications that use the DRDA protocol. With the DRDA protocol, data at the application server and application requester could be represented by different CCSIDs, as for example in the ASCII and EBCDIC environments. The CCSID clause consists of the keyword CCSID followed by an integer and assigns the integer identifier as the CCSID attribute of the column. This attribute defines the specific character set, code page, and encoding scheme used to represent the data in the character or graphic column.

For more information on column clauses and CCSIDs, see the *DB2 Server for VSE & VM SQL Reference* manual.

Storing Your Tables

When you create a table, the database manager inserts it into a section of the database that is reserved for you. For DB2 Server for VSE, these sections are called *dbspaces*. For DB2 Server for VM, these sections are called *private dbspaces*.

There are private dbspaces and public dbspaces. If a DBA has given you RESOURCE authority, you can acquire private dbspaces to contain your tables. You can also create your tables in public dbspaces if you have RESOURCE authority. If you do not have RESOURCE authority, you can create tables only if a DBA has acquired a private dbspace for you.

If you have more than one dbspace, you can let the database manager choose the one in which to place your tables, or you can specify the dbspace yourself. For example, assume that you have had two dbspaces reserved for you named *JOHN1* and *JOHN2*. To place the EMPL table in JOHN2, you type the following when you create the table:

```
create table empl -
(empno char(6), -
 lastname varchar(15), -
 edlevel smallint, -
 birthdate date, -
 quarter integer) -
in john2
```

Copying Data from Other Tables

Data can be loaded into your table by copying data from another table. This is performed using a variation of the INSERT statement. For example, load (copy) data from the EMPLOYEE table into the EMPL table using the following INSERT statement:

```
insert into empl (lastname,edlevel,birthdate,quarter) -
  select lastname,edlevel,birthdate,'014'  -
  from employee
```

This form of the INSERT statement uses a subquery instead of the VALUES clause. The information retrieved by the subquery is placed into the table as if multiple INSERT statements had been entered.

In the example above, the database manager is instructed to copy the employee information from each row of the EMPLOYEE table into the EMPL table. The database manager is also instructed to place the value *014* in the QUARTER column for each row inserted.

Dropping a Table

You can drop (delete) one of your tables using a DROP TABLE statement. For example, drop the table just created by typing:

```
drop table empl
```

The DROP TABLE statement deletes all the rows of the table, as well as the definition of the table. (Recall that the DELETE statement, discussed in "Deleting Table Data" on page 79, deletes individual rows.)

Identifying the Minimum Contents of a Table

When you create a table, you can identify which columns must contain entries. You would typically want key fields such as EMPNO in the EMPLOYEE table to always contain valid (not null) values. You prevent the use of nulls by specifying the NOT NULL option in the column definitions of CREATE TABLE statements.

For example, the following statement can be used to prevent nulls from being used for the EMPNO and LASTNAME columns of the EMPL table:

```
create table empl -
(empno char(6) not null, -
 lastname varchar(15) not null, -
 edlevel smallint, -
 birthdate date, -
 quarter integer)
```

Specifying NOT NULL can be very useful in copying data from other tables because it prevents incomplete rows from being inserted into your table.

In the above example, the BIRTHDATE column was defined as NOT NULL, because nulls are allowed for the corresponding column in the EMPLOYEE table.

Adding a Column to a Table

When you create a table, it is not necessary to know or specify all columns. Additional columns can be added later using the SQL statement ALTER TABLE. For example, you can add a column to the EMPL table to contain employee salary information by typing:

```
alter table empl -
add salary decimal(9,2)
```

You cannot specify the NOT NULL option in an ALTER TABLE statement. All existing rows of the table assume a null value for the SALARY column as a result of the ALTER TABLE statement.

After a column has been added, you can insert values using the UPDATE statement. See “Updating Rows” on page 75.

Specifying Referential Constraints

Referential constraints can be specified when tables are defined, or they can be added later.

When a primary key is added to an existing table, the database manager checks the table to ensure that all keys are unique. When a foreign key is added, the database manager checks all non-null foreign keys to ensure that they exist in the parent table.

Constraints can also be dropped, activated, or deactivated. When referential constraints have been deactivated, the database manager suspends checking, and the tables become unavailable for access by anyone other than the owner of the table or by someone possessing DBA authority.

Tables are deactivated, for example, to load large amounts of data onto the table. Since checking of the data has been suspended, the speed of the loading process increases considerably. When loading is complete, the table is activated.

When the primary key is deactivated, the primary key index on the parent table is automatically dropped and all active dependent foreign keys are implicitly deactivated. When a primary key is deactivated, all associated foreign keys are implicitly deactivated. When a primary key or a dependent foreign key is deactivated, all tables involved in the referential constraint become unavailable until the keys are activated once again. Activating the keys causes the database manager to validate the references in the data, and referential constraints are automatically enforced once again.

Additional information is provided for activation and deactivation in “Activating and Deactivating Primary Keys, Foreign Keys, or Unique Constraints” on page 185.

You can remove a referential constraint by dropping the foreign key. Dropping a table containing foreign keys removes the constraints associated with its keys. When a table containing a primary key is dropped, any foreign keys that reference the primary key are dropped automatically, thereby removing all the constraints that reference the primary key.

Identifying Required Privileges

The following table identifies the privileges required for changes to the referential structure.

Figure 112. Privileges Required

ALTER TABLE Clause	Privilege on Parent Table	Privilege on Dependent Table
Add Column	ALTER	
Add Primary Key	ALTER	
Add Foreign Key	REFERENCES	ALTER
Drop Primary Key	ALTER REFERENCES (1)	ALTER
Drop Foreign Key	REFERENCES	ALTER
Deactivate Primary Key	ALTER REFERENCES (1)	ALTER
Deactivate Foreign Key	REFERENCES	ALTER
Activate Primary Key	ALTER REFERENCES (1)	REFERENCES
Activate Foreign Key	REFERENCES	REFERENCES[2]
Create Foreign Key	REFERENCES	not applicable
Create Primary Key	not applicable	not applicable

Notes:

1. The REFERENCES privilege is required only if the parent table has any dependents.
2. The ALTER privilege is required if the primary key has any foreign keys defined on it.

To add, drop, deactivate, activate, or create a unique constraint, you must have the ALTER privilege on the table. See the discussions of the CREATE TABLE and ALTER TABLE statements in the *DB2 Server for VSE & VM SQL Reference* manual.

Creating a Table That Contains a Primary Key

You can create a table with a primary key by using a PRIMARY KEY clause in the CREATE TABLE statement. This clause specifies the column that is the primary key. For example, create a new table for students that contains a student first name, last name, and student number (the primary key) with the following statement:

```
create table students -
(firstname varchar(12)    not null, -
 lastname  varchar(15)    not null, -
 studentno char(6)        not null, -
 primary key (studentno))
```

The last line in the CREATE TABLE statement defines the STUDENTNO column as the primary key for this table. A column named as a primary key must have been defined with the NOT NULL option.

Adding a Primary Key to an Existing Table

It is not necessary to define the primary key in the CREATE TABLE statement. It can be added later using the ALTER TABLE statement. You can create the table first using:

```
create table students -
(firstname varchar(12)    not null, -
 lastname varchar(15)     not null, -
 studentno char(6)        not null)
```

Then add a primary key:

```
alter table students -
add primary key (studentno)
```

This makes the STUDENTNO column the primary key in the STUDENTS table if there are no duplicate values in that column. If duplicate values exist when you attempt to add a primary key on an existing column, the ALTER TABLE statement fails. If the column named for the primary key allows nulls, the statement also fails.

Creating a Table That Contains a Foreign Key

You can create a table with a foreign key by adding the FOREIGN KEY clause to the CREATE TABLE statement. This clause specifies the column that will be the foreign key and the table containing the primary key to be referenced. The parent table referenced must already exist and must have a primary key defined.

Create a table for a computer science class that contains a row for each student enrolled in the class and references the STUDENTS table as follows:

```
create table cs110 -
(studentno char(6) not null, -
 midterm   integer, -
 final     integer, -
 foreign key r_studt (studentno) references -
           students on delete cascade)
```

This creates a table where every row must represent a student who is listed in the STUDENT table. If a student is deleted from the STUDENT table, that student is also automatically deleted from this class list because of the delete cascade rule specified in the foreign key definition.

The referential constraint defined in the above example is `r_studt`. This name is used when the foreign key is deactivated, activated, or dropped.

Adding a Foreign Key to an Existing Table

Foreign keys can be added after a table has been created by using the ALTER TABLE statement. The dependent table CS110 can be created by creating the table first and without the foreign key:

```
create table cs110 -
(studentno char(6) not null, -
 midterm   integer, -
 final     integer)
```

You then add the foreign key using the following statement:

```
alter table cs110 add -
foreign key r_studt (studentno) references -
            students on delete cascade
```

When a foreign key is added in this way, all foreign key values currently in the table must match existing values in the primary key referenced, or the attempt to add a foreign key fails.

When creating referential constraints involving two tables that reference each other (such as the EMPLOYEE table and the DEPARTMENT table in the referential structure in Figure 119 on page 269), at least one of the foreign keys must be added after the table has been created. It is impossible to reference a table (and its primary key) if that table has not been created. To create this type of structure, create one table with its primary key. Create the second table with its primary key and the foreign key referencing the first table. Then, add a foreign key to the first table which references the primary key in the second. See the create statements for the EMPLOYEE and DEPARTMENT tables in Appendix B, "Sample Tables" on page 269 for examples of how this can be performed.

Activating and Deactivating Primary Keys, Foreign Keys, or Unique Constraints

The constraints placed on altering tables that contain primary or foreign keys, or unique constraints, can be suspended by deactivating the keys in the table. For example, the primary key in the STUDENTS table is deactivated with the following statement:

```
alter table students deactivate primary key
```

The above statement causes the restrictions on inserting, deleting, and updating to be suspended until the key is reactivated. No other users are allowed access to a table while it has an inactive key. In addition, keys that are related to an inactive key through a referential constraint are also considered inactive by the database manager. If the primary key in STUDENTS is deactivated, the foreign key in CS110 becomes inactive, and that table cannot be accessed.

To activate an inactive key, you must alter the table as follows:

```
alter table students activate primary key
```

If you make changes to the STUDENTS table while its primary key is inactive, the result of those changes cannot violate any of the constraints on the primary key, or of the referential constraint. If the changes produced any duplicate primary key values, dependent foreign key values without matching primary key values, or null primary key values, the primary key activation fails.

Foreign keys can be deactivated and then activated in the same way as primary keys by using an ALTER TABLE statement. When activating or deactivating a foreign key, the ALTER TABLE statement must include the name of the referential constraint. Deactivating the foreign key in the referential constraint *r_studt* for the CS110 table is accomplished by typing:

```
alter table cs110 deactivate foreign key r_studt
```

You can also use the ALTER TABLE statement to deactivate and activate unique constraints. As with a foreign key, you have to use the constraint name when you deactivate or activate it. To deactivate the unique constraint *empno* for the table TEACHERS, type:

```
alter table teachers deactivate unique empno
```

This statement causes the restrictions on inserting and updating to be suspended until the constraint is reactivated. For more information about adding a unique constraint to an existing table, or creating a table that contains a unique constraint, refer to the *DB2 Server for VSE Database Administration* and *DB2 Server for VM Database Administration* manuals.

Determining Effects on Stored Format Information

Performing management tasks on your tables also affects any stored queries you have. The result of these tasks depends on the type of query stored and the type of changes made to the table.

If the table referred to by a stored query is changed by DROP TABLE, CREATE TABLE, or ALTER TABLE statements, the formatting information stored with that query may no longer be valid. For example, suppose a stored query performed grouping on columns 1 and 2 of a table. These two table columns are defined as VARCHAR.

The table is dropped and recreated with column 1 now defined as CHAR and column 2 again defined as VARCHAR. The formatting information saved in the stored query for column 1 is no longer valid. The formatting for the other columns is still valid if the data types defined for these columns is the same in the recreated table as in the original table.

In this last example, you get the same result if the stored query did not contain a GROUP BY clause.

Sharing Your Tables with Other Users

When you create a table, you become its owner. Only you and a person with DBA authority can use the table. However, you may want to share access to your tables. This section describes the sharing of your data with other users, including the determination of what data they can use, and what they can do with it.

You give access to your data using an SQL GRANT statement, and you take away access using the SQL REVOKE statement. These statements and other techniques for sharing data are discussed in the following sections.

Granting the Privilege to Select Data from Your Tables

The most common reason for sharing data is to let another user query your tables.

To illustrate, here is an example of how you would grant select capability on your EMPL table to user Farley:

```
grant select -
on empl -
to farley
```

If you want the recipient of the privilege to be able to grant that privilege to other users, add the keyword phrase WITH GRANT OPTION at the end of the statement:

```
grant select -
on empl -
to farley -
with grant option
```

Granting INSERT, UPDATE, and DELETE Privileges on Your Tables

These privileges are also granted using the GRANT statement and can be included in a single GRANT statement. For example, grant INSERT and UPDATE capability on the EMPL table to Farley by typing:

```
grant insert,update -
on empl -
to farley
```

Now Farley can insert, update, and query EMPL. Note, however, that because you did not identify the DELETE function in either of the GRANT statements, Farley cannot delete rows.

Note also that when you grant *multiple* functions, they are separated in the statement by commas.

Granting the Privilege to Reference a Primary Key

You can allow users to reference the primary key in a table you have created. Users can have access to the data in your table through a GRANT SELECT but they may not reference your table in a referential constraint without the REFERENCES privilege. Grant the REFERENCES privilege to Farley using the following statement:

```
grant references -
on students -
to farley
```

This privilege allows Farley to define a referential constraint for a table being created or for an existing table that references STUDENTS as the parent table.

Granting the Privilege to Change Columns or Keys

You can allow other users either to add columns, a primary key, foreign keys, or a unique constraint to a table, or to drop a primary key, foreign keys or a unique constraint from a table. Grant this privilege on the CS110 table to Farley using the following statement:

```
grant alter -
on cs110 -
to farley
```

The ALTER privilege is required to change the state of a referential constraint on a table only if you are not the owner of that table. With this privilege, you can add a referential constraint (and a foreign key), drop a referential constraint, deactivate an existing primary or foreign key, or activate an inactive primary or foreign key. You also have the privilege of manipulating unique constraints.

Note: Your ability to change the state of referential constraints depends on both the ALTER and the REFERENCES privileges you have for the two tables involved. For example, you cannot add a referential constraint unless you have the ALTER privilege for the dependent table and the REFERENCES privilege for the parent table. You require the ALTER privilege for both parent and dependent tables if you want to drop a referential constraint.

Granting All Privileges to a User

You can grant all privileges that you have on the table by specifying ALL instead of listing all the privileges individually. For example, if you want to grant all your privileges on table EMPL to user Marcus, you type:

```
grant all -
on empl -
to marcus
```

This statement permits Marcus to perform the same grantable SQL operations on your EMPL table as you can. It should be noted, however, that certain capabilities are reserved for the owner of the table. One such capability is the DROP TABLE function. The complete list of grantable SQL functions is shown in Figure 113.

Figure 113. Grantable Privileges

Grantable Privilege	**Capability Granted**
ALTER	This lets another user add columns to the table or add, drop, deactivate, or activate primary keys, foreign keys, and unique constraints. To add a foreign key, you also need the REFERENCES privilege on the parent table.
DELETE	This lets another user delete rows from the table.
INDEX	This lets another user create an index on the table.
INSERT	This lets another user add rows to the table.
REFERENCES	This lets another user reference this parent table (table for which REFERENCES is granted) when a foreign key is added, dropped, activated, or deactivated. To alter a foreign key, you also need the ALTER privilege on the dependent table.
SELECT	This lets another user see data in the table.
UPDATE	This lets another user update columns in the table.
ALL PRIVILEGES or ALL	This lets another user have all the privileges that you have on the table.

Restricting the Update Privilege to Certain Columns

You may want to give a user responsibility for maintaining only certain types of information in your table. That is, you want to give that person update capability only on selected columns.

For example, if you wanted user Mona to keep the QUARTER column in the EMPL table updated, you give her the authority by typing:

```
grant select,update(quarter) -
on empl -
to mona
```

You can grant update capability to multiple columns by including all the columns within the parentheses and separating them with commas.

Granting Privileges to Multiple Users

If you want to grant the identical capabilities to a *group* of users, you use a single GRANT statement. For example, assume Mona is part of a department responsible for maintaining the quarter information. Users Jim, Dan, and Dee also need update privileges on the QUARTER column of the EMPL table. You can extend the update capability to all three by typing:

```
grant update(quarter) -
on empl -
to jim,dan,dee
```

You can also grant privileges to the public within a list of grantees as illustrated below:

```
grant update(quarter) -
on empl -
to jim,public,dee
```

Revoking Granted Privileges

You can revoke any privilege that was assigned with the GRANT statement. For example, if Dan changes jobs and is no longer responsible for maintaining quarter information, you can revoke his privileges on the EMPL table by typing:

```
revoke update -
on empl -
from dan
```

Note that Dan's entire (all table columns) update privilege is revoked. To revoke a privilege on a specific column, you have to revoke that privilege entirely and then grant it for the columns on which the privilege is to be retained.

Multiple users and privileges can be identified in a single REVOKE statement. The ALL option can be used to revoke all privileges you have granted on a table (in the same manner as used in the GRANT statement). You can include *public* only once in a list of authorization names in a REVOKE statement.

Using a View to Restrict Privileges to Certain Rows

Views can be used to restrict a privilege to specific rows of your tables. For example, assume Jim and Dee maintain quarter information for different groups of employees based on level of education. The following view supplies the information Jim needs for employees of education level 16 or less:

```
create view to16 -
as select empno,lastname,edlevel,birthdate,quarter -
   from empl -
   where edlevel <= 16
```

Similarly, the following view supplies all the information needed by Dee:

```
create view past16 -
as select empno,lastname,edlevel,birthdate,quarter -
   from empl -
   where edlevel > 16
```

The next step is to grant Jim and Dee privileges on the views. Type the following GRANT statements:

```
grant select,update(quarter) -
on to16 -
to jim

grant select,update(quarter) -
on past16 -
to dee
```

Using a View to Restrict Privileges to Certain Columns

Views can also be used to restrict the columns on which a user can type SELECT or INSERT statements. For example, you can give Jim and Dee the capability to select salary information from the EMPLOYEE table, but restrict them from viewing commission information simply by specifying the SALARY column and omitting the COMM column when you create the view. Create the view using:

```
create view blindempl -
as select salary -
from employee
```

Now give Jim and Dee table privileges:

```
grant select -
on blindempl -
to jim,dee
```

Creating Tables That You Want to Share

In an earlier section of this chapter, a private dbspace as a section of the database reserved for your tables was described. A private dbspace is suitable for your personal tables and is not really appropriate for tables that are to be shared.

Instead, a public dbspace should be used. When a table is inserted into a public dbspace, multiple users can update the table simultaneously. No two users, however, can use *the same row* at the same time.

To insert a table into a public dbspace, provide the database manager with the name of the dbspace using the IN clause of the CREATE TABLE statement. For example, the EMPL table can be created in a public dbspace named *SAMPLEDB* by typing:

```
create table empl -
(empno char(6), -
 lastname varchar(15), -
 edlevel smallint, -
 birthdate date, -
 quarter integer) -
in sampledb
```

When the dbspace name is not supplied on the CREATE TABLE statement, the table is created in one of your private dbspaces.

Accessing Tables Belonging to Other Users

When you refer to a table (or view) in an SQL statement, the database manager assumes you are referring to a table that you own. If you want to access a table that belongs to another user, you must identify the user and the table name. This is done by inserting the owner's authorization ID before the table name and separating the two with a period.

For example, you can access the system copy of the EMPLOYEE table (created during the installation of the database manager) by using the following query:

```
select * -
from sqldba.employee
```

This query retrieves the copy of the EMPLOYEE table that is owned by the sample user, SQLDBA. Your personal copy of the EMPLOYEE table is ignored. Similarly, other users have to use this method to access your EMPL table.

If an authorization ID does not begin with a letter, number, $, #, or @, you must enclose it in double quotation marks. For example:

```
select * -
from "%A23C".payroll
```

For detailed information about identifier naming conventions, see the *DB2 Server for VSE & VM SQL Reference* manual.

Using Synonyms

To avoid specifying an authorization ID for another user's table or view that you access frequently, you can create *synonyms* for their tables and views. For example, you can assign a synonym to be used in place of SQLDBA.EMPLOYEE by typing:

```
create synonym dbaemp -
for sqldba.employee
```

Now you can refer to SQLDBA.EMPLOYEE by the synonym DBAEMP. For example, if you type:

```
select * -
from dbaemp
```

The database manager displays all information from the EMPLOYEE table.

You can also use the CREATE SYNONYM statement to assign a synonym to one of your own tables. For example, if you are user JOHN, you can assign the synonym *ih* to your EMPL table by typing:

```
create synonym ih -
for john.empl
```

When you finish using the table or view for which you have defined the synonym, you should drop the synonym. To drop the synonym *ih*, for example, type:

```
drop synonym ih
```

The table or view on which the synonym was defined is not affected by this command.

Note: Because the performance of a DROP TABLE or DROP VIEW statement does not drop associated synonyms, you must drop the synonym(s) yourself.

Improving Query Performance

There are several ways you can improve data-access performance. This section describes three methods: indexing a table, updating statistics, and locking data.

Indexing a Table

The database manager uses an *index* to locate particular rows of a table.

Although you create these indexes, you do not use them directly. You simply enter your query and the database manager searches for, and attempts to use, an appropriate index to locate the information. The database manager finds the information whether an index exists or not, but it may find the information faster using an index.

A good table index is one that anticipates the kinds of queries to be used for the table.

For example, if you typically look for departments in the DEPARTMENT table by department name, create an index for the DEPTNAME column by typing the following SQL statement:

```
create index dptnme -
on department -
(deptname)
```

You can delete an index by using the DROP INDEX statement. For example, drop the DPTNME index by typing:

```
drop index dptnme
```

You can create a unique index when you create a table by using the CREATE TABLE statement, or you can add a unique index to an already existing table by using the ALTER TABLE statement. For more information about the UNIQUE attribute of these statements, see the *DB2 Server for VSE & VM SQL Reference* manual.

A unique index is also automatically created for each primary key and dropped automatically when the primary key is dropped. The primary key is dropped either by using an ALTER TABLE statement or by dropping the table or dbspace.

You cannot create a unique index on a VARCHAR or VARGRAPHIC column that has values that differ only by trailing blanks. Trailing blanks are ignored for values with these data types. Therefore, 'Adm ' is the same as 'Adm'.

If a VARCHAR or VARGRAPHIC column is not defined as unique, the trailing blanks on values do not affect the index. The order of 'Adm ' and 'Adm' is unpredictable because they only differ in trailing blanks.

Maintaining Updated Statistics

Another performance consideration concerns data statistics that are kept in the database manager catalogs. These statistics provide information about tables such as the number of rows in a particular table, or the number of different values contained in a particular column. This information is used by the database manager to determine the best method to satisfy query requests.

It may be important to keep these statistics up to date. For example, some tables are rarely updated, and their statistics change very little over the life of the table. Other tables, however, are updated frequently, and their statistics should be updated periodically.

You can update the statistics on a table by using an SQL UPDATE STATISTICS statement. For example, update the statistics on the ACTIVITY table by typing:

```
update statistics -
for table activity
```

Updating statistics involves a scan of both the rows and indexes of a table. This can be time-consuming for a large table; performing the update during off-peak hours is a good idea. As a general rule, try to develop a rule of thumb for determining when to update table statistics; for example, you might decide to update a table when it has changed by 20% or more. The full description of the UPDATE STATISTICS statement is shown in the *DB2 Server for VSE & VM SQL Reference* manual.

Locking Data

When you update or delete data in a table, the table is considered unstable because its data is changing. The database manager protects you and other users from obtaining unreliable query results by limiting access to unstable tables. It also prevents deadlocks when several users are trying to update the same data.

The database manager isolates the data by *locking* it. You can control the amount of data locked from other users, as well as the length of time that the lock is applied. Controlling the amount of locked data affects system performance. For example, a small amount of locked data requires less processing time than a large amount.

If you try to access an SQL object while another user is locking it, your processing is suspended until the other user has finished with the object. If you do not want to wait indefinitely, you can type the following to stop your transaction:

```
cancel
```

The database manager then rolls back any uncommitted work, and issues messages ARI7043I and ARI7040I. You can find more information about the CANCEL command under “CANCEL” on page 206.

Specifying the Isolation Level

You control locks by specifying the *isolation level*. The isolation-level setting affects only those tables stored in public dbspaces.

The isolation level you set is related to the task you are performing. The selection, insertion, updating, and deletion of table data are all affected by the isolation level. Specifically, the isolation level determines *how soon* data you have read can be changed by other users.

The isolation level has three settings: *repeatable read* (RR), *cursor stability* (CS), and *uncommitted read* (UR).

You can use the SET ISOLATION command only when the target AS is a local application server. Otherwise, the isolation level of CS is used and the SET ISOLATION command will have no effect.

Using the Repeatable Read Setting: Use the RR setting when you are modifying data. The RR setting ensures that data is completely isolated for your use. No other user can update the data until your work has completed.

Using the Cursor Stability Setting: Use the CS setting when you are simply querying (selecting) committed data. The term *cursor* in this case refers to the database manager cursor that points to the data in the table that you are using. The data involved in a CS setting is unlocked as soon as possible by the database manager for other users.

Using the Uncommitted Read Setting: Use the UR setting when you are querying (selecting) either committed or uncommitted data. With this setting, data can be read without waiting for other logical units of work that are updating the data and reading data will not prevent other application processes from updating it. However, you should remember that data integrity may be compromised with this setting and that UR should only be used when it is not necessary that the data you are reading be committed.

Using the SET Command

You use the SET command to control the isolation level. The SET command format is given on page 245, and an explanation of isolation level settings is found on page 248. See Chapter 9, “Creating and Using Routines” on page 125 for information on how to set the isolation level from a routine.

Note: Do not forget to change the isolation level back to the default isolation level when you have completed the operation for which you changed it.

Handling Lock Contention

Use an RR setting unless lock contention is a significant problem on the application server you are accessing. If lock contention is significant, use a CS setting whenever you can. The UR setting minimizes lock contention but it should only be used when it is not necessary that the data you are reading be committed.

Determining the Isolation-Level Setting

Figure 114 and Figure 115 describe isolation level settings for various tasks. The first table discusses cursor stability, and the second, repeatable read. There is no table for the uncommitted read setting because this setting is not recommended for regular use.

Figure 114. Guidelines for Using Cursor Stability

Activity	Description
Browsing a query result	When you are simply displaying data, the isolation level cursor stability is sufficient. At this time, do not plan to update the table or to print a formal report.
Preparing sample reports	While you are preparing a draft of a report, which you are using to check format and general content, use isolation level cursor stability.
Selecting or printing read only data	Read-only data resides in tables that are subject to controlled maintenance (insert, update, or delete). That is, any maintenance is done on a known periodic basis (for example, tables that are only updated overnight).
Browsing HELP information	ISQL HELP text information is read-only data, and isolation level cursor stability is sufficient.
Using SQL data definition operations	Read and update access to a catalog table during the performance of data definition statements (CREATE, ACQUIRE, GRANT) is always done with an isolation level setting of repeatable read. You do not have to set the isolation level to protect your definitions.
Using EXPLAIN	Using the EXPLAIN statement to access a catalog table is always performed with isolation level repeatable read, regardless of your isolation level setting.
Accessing data in private dbspaces	Accessing private dbspaces is effectively isolation level repeatable read, because locking is only performed at the dbspace level. You do not need to adjust your setting for isolation level.
Accessing data in public dbspaces with dbspace level locking	For public dbspaces with dbspace locking, access is always effectively isolation level repeatable read, because locking is only performed at the dbspace level. You need not change your isolation level setting.
Working with stored queries	Stored queries are always stored and recalled with an isolation level repeatable read. You control the isolation level used for starting a stored query. Therefore, you must set the isolation level to the desired value before starting the stored query.
Invoking routines	Retrieving from a routine is performed with an isolation level repeatable read. You control the isolation level used for running a routine. You can change the isolation level setting before running the routine or within the routine (as many times as needed).
Using ISQL commands	For ISQL commands that access the application server, the isolation level used is repeatable read. For the ISQL commands RUN, START, and HELP, you can control the setting of the isolation level.
Using the INSERT statement and the INPUT command	Using INSERT statements with values and INPUT commands are not affected by the isolation level setting because no data is read from the application server. Using INSERT statements with subselect are affected by the isolation level setting because of the SELECT clause. You must follow the guidelines for selecting data given above when you use INSERT through subselect statements.

Figure 115. Guidelines for Using Repeatable Read

Activity	Description
Using DELETE, INSERT, or UPDATE from a display	If you type a DELETE, INSERT, or UPDATE statement based on a SELECT statement result, do not set the isolation level to repeatable read before you type the SELECT statement. It is then impossible for the displayed data to change before you have typed the DELETE, INSERT, or UPDATE statement. **Note:** In addition, set AUTOCOMMIT OFF so the SELECT and data change statements are contained in the same LUW. You need not set AUTOCOMMIT OFF and isolation level repeatable read if any of the following is true: • Data selected is read-only. • You are the only person authorized to modify the selected data. • You have other ways of ensuring the data selected does not change. • A command that changes the contents of the table is valid even if the selected data does change.
Using DELETE or UPDATE statements	Use an isolation level repeatable read when you delete or update data unless: • You are the only person authorized to modify the data. • You have other ways of ensuring the data selected does not change.
Preparing formal reports	To prevent data from changing, set the isolation level to repeatable read when you prepare a formal report unless: • Data selected is read-only. • You are the only person authorized to modify the selected data. • You have other ways of ensuring the data selected does not change.

EXERCISE 16

(Answers are in Appendix A, Answers to the Exercises, on page 268.)

Perform the following:

1. Create a table that has the following characteristics:

   Table Name

   `OLD_ACTIVITY`

   Column Names

   ```
   ANUMBER  - a number column containing numbers from 1 to 999, do
              not allow nulls.
   AKEYWORD - a varying character (max=10) column, do not allow nulls.
   ADESCR   - a varying character (max=30) column, do not allow nulls.
   ```

2. Fill the table with the contents of the ACTIVITY tables owned by the system (SQLDBA).
3. Add a fourth column to this table (call it STATUS) having a varying character length not exceeding 8 characters.
4. Grant the update privilege, on the STATUS column only, to Mona and Lisa.
5. Grant the select privilege for those rows that contain *inactive* in the STATUS column to Farley.

Chapter 12. Using VM Functions

In this chapter, you progress beyond ISQL and learn some VM commands and functions that can enhance your use of the database manager. Although this chapter does not show you how to use VM, it does identify many of the VM features that help you get the most out of ISQL and this RDBMS.

The commands described in this chapter are neither SQL statements nor ISQL commands; they are CMS and CP commands that can be used to control the characteristics of your virtual machine, direct printed output, and provide additional VM facilities. As CMS and CP commands, they can only be used in CMS or CMS subset mode.

CMS-Subset Processing

During your ISQL session, you can enter CMS or CP commands without terminating your ISQL session. When finished you can return to ISQL.

To enter CMS-subset mode, type:

```
cms
```

You can now type CMS or CP commands, EXEC procedures, or user programs.

Note: Do not enter any EXEC procedure or program that accesses the application server while you are in CMS subset mode.

Returning to ISQL from CMS Subset Mode

When you finish entering your CMS and CP commands and are ready to return to ISQL, type:

```
return
```

Entering CP Commands

You can enter CP commands in three ways:

- A CP command can be typed at any time during ISQL processing by prefixing the command with *#CP*. This procedure immediately interrupts the current ISQL processing, enters CMS subset mode and performs the designated CP command, and then returns to ISQL when the command is complete.
- You can press PA1 at any time. `CP READ` appears in the status area and you can type any CP commands.

  If you are running with CP SET RUN ON, control automatically returns to ISQL after each CP command is executed. ISQL resumes processing from the point of interruption. If you are running with CP SET RUN OFF, you must type a B or begin on the input line and press ENTER to resume processing in ISQL.

  The SET FULLSCREEN ON command cancels the interrupt action of PA1. Suspending the full-screen option or turning it off does not reset PA1 to the original interrupt setting. If PA1 does not interrupt ISQL, you can reestablish it as an interrupt key with the following command:

  ```
  #cp terminal brkkey pa1
  ```


- While in CMS subset mode, you can type CP commands (with or without the CP prefix) in the same manner as that described above for CMS commands.

As a DB2 Server for VM user, you probably use CP commands only when you want to change printer spooling or routing characteristics, or change PF key definitions.

Obtaining Printed Reports on a Workstation Printer

Your printed output is automatically sent to the printer designated by your site. You can change or redirect your printed output to another printer and specify other print characteristics by using the CP commands TAG and SPOOL. The TAG command identifies the receiving destination; the SPOOL command directs the printed output to the network machine.

First, start the statement stored as DEPT in Chapter 8, “Storing SQL Statements” on page 119 by typing:

```
start dept
```

Suppose that you want to print this information on a remote printer. You have to know the node ID for that printer and the user ID of the network machine. You can use the CMS IDENTIFY command for information about network identifiers (NETIDs). For this example, assume that you want to send your output to a 3262-13 printer with a destination ID of RMT3262. Enter CMS subset mode by typing the following:

```
cms
```

Then type the following CMS command:

```
identify
```

The results of this command give you the necessary information to use the SPOOL and TAG commands. The format of the output produced by typing this command is:

```
user AT nodeid VIA netid
```

Now you would type the CP SPOOL and TAG commands to route your printed output to the 3262 printer:

```
#cp spool printer to netid nohold
#cp tag dev printer RMT3262 system
```

To return to ISQL and print your report, type:

```
return
print
```

Specifying the Number of Copies of Printed Reports

You can use the CP SPOOL command to specify the number of copies for *all* reports you print during the current terminal session.

For example, you can specify three copies by modifying the previous CP SPOOL command example as follows:

```
#cp spool printer to netid copy 3
```

All following PRINT commands use the quantity specified by the most recently typed CP SPOOL command or ISQL PRINT command with the COPIES keyword.

Note: This section assumes your site is running with the Remote Spooling Communications Subsystem (RSCS) Networking Program Product (Release 3 5748-XP1, or Version 2 Release 1 5664-188).

Using EXEC Files

An EXEC is a file with a file type of EXEC. It contains a series of commands and statements that are executed when the file name of the EXEC file is typed.

You can use EXECs to stack SQL statements and ISQL commands before you begin your ISQL session. As soon as you start ISQL, this information is read from the stack and executed. By using EXEC processing in this way, you can predefine all default settings to be used during your display session, establish PF key values, and designate print specifications. This method can also be used to automatically start procedures tailored to the needs of specific users.

The EXEC examples in this chapter do not include the CONNECT statement as part of the stacked set; it is assumed that you are working with your own sample tables. If you must gain access to another user ID to use ISQL, use a CONNECT statement as the first stacked statement. See the *DB2 Server for VSE & VM SQL Reference* manual for additional information on the CONNECT statement.

For information on creating CMS EXEC files, see the *VM/ESA: CMS User's Guide* manual. For information on the REXX language, see the *VM/ESA REXX/VM User's Guide* and the *VM/ESA REXX/VM Reference* manuals.

Stacking Commands in an EXEC File

In the CMS environment, you can run EXECs that stack SQL statements and ISQL commands to be processed automatically as soon as ISQL begins. If the EXEC also starts ISQL, the user of the EXEC need not know anything about ISQL.

You can write specialized EXECs for your own use or for other users. Figure 116 shows an EXEC that is written in the REXX language:

```
/* this exec develops a report for project mean numbers */
   Queue 'SELECT * FROM PROJ_ACT ORDER BY PROJNO'
   Queue 'FORMAT GROUP PROJNO'
   Queue 'FORMAT SUBTOTAL ACSTAFF'
   Queue "FORMAT TTITLE 'PROJECT MEAN EMPLOYEES'"
   Queue 'PRINT'
   Queue 'END'
   Queue 'EXIT'
'EXEC ISQL'
Exit   /* end of exec */
```

Figure 116. Example of Stacked ISQL Commands in a REXX EXEC

Prompting by Using an EXEC File

To create a prompt environment for a casual SQL user, you can write an EXEC like the one shown in Figure 117. The user would run the EXEC from the CMS environment without starting ISQL.

```
/* This exec prompts for a table to be viewed or printed */
Trace Value 'OFF'
Say 'WHICH TABLE WOULD YOU LIKE TO SEE?'
Parse Upper Pull tablename
Say 'WOULD YOU LIKE TO HAVE THIS TABLE PRINTED? (Y OR N)'
Parse Upper Pull printoption
Say 'YOU MUST ENTER END TO LEAVE THE DISPLAY OF THE TABLE'
If printoption = 'Y' then Do
   Queue 'SELECT * FROM' tablename
   Queue 'DISPLAY'
   Queue 'PRINT'
   Queue 'END'
End
Else If printoption = 'N' then Do
   Queue 'SELECT * FROM' tablename
   Queue 'DISPLAY'
   Queue 'END'
End
If printoption = 'Y' | printoption = 'N' then Do
   Queue 'EXIT'
   'EXEC ISQL'
End
Exit /* End of exec */
```

Figure 117. Example of an EXEC That Prompts the User

When you run this EXEC, the following user/system dialog occurs. If the EXEC has a file name of MYTABLES, you type:

mytables

The system prompts you with the message `WHICH TABLE WOULD YOU LIKE TO SEE?`. You type:

department

The system prompts you with the message `WOULD YOU LIKE TO HAVE THIS TABLE PRINTED? (Y OR N)`. You type:

y

The system prompts you with the message `YOU MUST ENTER END TO LEAVE THE DISPLAY OF THE TABLE`.

The ISQL startup messages immediately follow. Then the partial display in Figure 118 on page 201 should appear.

```
DEPTNO  DEPTNAME             MGRNO   ADMRDEPT
------  --------------------  ------  --------
A00     SPIFFY COMPUTER SERV< 000010  A00
B01     PLANNING              000020  A00
C01     INFORMATION CENTER    000030  A00
D01     DEVELOPMENT CENTER    ?       A00
D11     MANUFACTURING SYSTEM< 000060  D01
D21     ADMINISTRATION SYSTE< 000070  D01
E01     SUPPORT SERVICES      000050  A00
E11     OPERATIONS            000090  E01
E21     SOFTWARE SUPPORT      000100  E01
* End of Result *** 9 Rows Displayed ***Cost Estimate is 1*********************
```

Figure 118. Query Result from an EXEC

You see the ISQL signoff messages when you end the display.

Starting ISQL from a Terminal

Because ISQL is a full-screen interactive program, it must be started from a display terminal. When a user is automatically logged on using the CP AUTOLOG command, or when a user tries to start ISQL while disconnected, ISQL is not started. The user must be logged on to a display terminal to start ISQL.

Disconnecting after Starting ISQL

Although the restriction of a logged-on display terminal applies to the starting of ISQL, it does not apply if a user disconnects *after* starting ISQL. For example, if you have an EXEC that starts ISQL and then executes a number of ISQL SELECT statements, you can disconnect your display terminal after ISQL is started and the remainder of your SELECT statements are executed.

Avoid using commands and statements that result in the issuance of ISQL decision-type messages. For example, stacking a SET AUTOCOMMIT OFF command causes the following message to be issued:

```
ARI7602D  You are in a logical unit of work. Type COMMIT to
          have a COMMIT issued for you or ROLLBACK to
          have a ROLLBACK issued for you.
```

Your response to this message cannot be accepted from the command stack because of the interactive design of ISQL. You must type the response yourself.

Chapter 13. ISQL Commands

This chapter contains syntax diagrams, semantic descriptions, rules, and situations where you would use ISQL or operator commands. The commands are organized alphabetically.

BACKOUT

```
►►──BACKOUT──────────────────────────────────────────────────────────────────►◄
```

BACKOUT is an ISQL command that is meaningful only while you are using the INPUT command with AUTOCOMMIT on. It nullifies all changes made since the last SAVE command. If no SAVE command has been issued, all changes are nullified since the start of the INPUT command.

With AUTOCOMMIT off, BACKOUT has no effect. You must enter a ROLLBACK statement to nullify all input data. This also nullifies all changes made during the current logical unit of work. You must end the INPUT command (by using the END command) before issuing the ROLLBACK statement.

BACKWARD

```
►►──BACKWARD──┬───────────────┬────────────────────────────────────────────►◄
              ├─rows_integer──┤
              └─MAX───────────┘
```

This ISQL display command displays the rows above or before those currently being displayed.

rows_integer
is a number that indicates the number of rows backward that you want to move the display.

MAX
moves the display to the beginning of the query result.

If neither *rows_integer* nor MAX is specified, the display moves backward half a display.

Moving backward through the query result is limited: you can go backward to a limit of one full display from the last FORWARD command, or you can return to the beginning of the query result by issuing BACKWARD MAX.

Scrolling backward a half display at a time can be done by pressing PF7 (or PF19). Because you can reset your PF keys, you can assign the BACKWARD function to any PF key. The default function key is PF7 (or PF19).

Note: You cannot receive a prompt response when you enter a BACKWARD MAX command, because the query must be reexecuted to move the display back to the first display.

CANCEL

```
►►──CANCEL──────────────────────────────────────────────────────────────►◄
```

CANCEL is an ISQL command that you can use to cancel a command, an SQL statement, or logical unit of work in progress. CANCEL can be typed anytime you are entering data, commands, or statements.

DB2 Server for VSE

Canceling a command or statement also causes a ROLLBACK to be processed.

DB2 Server for VM

Canceling a command or statement also causes a ROLLBACK RELEASE to be processed. If you have previously issued an explicit CONNECT command, you must reconnect to the database manager.

If AUTOCOMMIT is on and the previous statement was an SQL INSERT, UPDATE, or DELETE statement that affected more than one row, all changes made by the previous statement are rolled back.

If AUTOCOMMIT is off, all work since the last COMMIT, or since the beginning of the logical unit of work performed by the following, is rolled back:

- SQL statements
- ISQL commands that cause changes to table data, stored SQL statements, or routines.

DB2 Server for VSE

Except for long-running SQL statements, to start a cancel operation, type:

```
cancel
```

DB2 Server for VM

To start a cancel operation, type:

```
cancel
```

When canceling a command with AUTOCOMMIT off, you are prompted to verify if a CANCEL should be performed. If you type NO, the CANCEL processing is not performed. If you answer YES, all changes made to the data since the last COMMIT, or since the start of the logical unit of work if there was no COMMIT, are rolled back.

When you cancel an ISQL INPUT command with AUTOCOMMIT on, all changes since the last SAVE command, or since the start of the INPUT command if there was no SAVE command, are rolled back.

DB2 Server for VSE

Long-running SQL statements are those for which message ARI7044I is issued to tell you that the terminal is free. These statements are cancelled by pressing CLEAR and typing the CANCEL command prefixed by the ISQL transaction identifier, ISQL. (If your location has redefined the ISQL transaction identifier, prefix the CANCEL command with the transaction identifier that your location has defined.)

The command looks like:

```
isql cancel
```

When canceling long-running SQL statements, a ROLLBACK is always performed.

CHANGE

```
►►──CHAnge──/──replaced_string──/──┬──────────────────────┬──────────────────────────►◄
                                   └─replacing_string──/──┘
```

CHANGE is an ISQL command that modifies the current SQL statement in the SQL command buffer and displays the results. If data in the SELECT or FROM clauses of a SELECT statement is changed, associated formatting information for that statement is erased. However, if the changed information is contained in the WHERE, GROUP BY, ORDER BY, or HAVING clauses, associated formatting information for the statement is saved.

/
: is any non-blank character that identifies the beginning and end of a string. The slash is a good choice unless you are changing data that contains a slash. This character must be separated from the command name by at least one blank and must not occur in either string.

replaced_string
: is the characters to be replaced in the current SQL statement. The string can contain DBCS characters.

replacing_string
: is the characters to replace the first occurrence of the characters in *replaced_string*. If *replacing_string* is omitted, the first occurrence of replaced_string is deleted. The string can contain DBCS characters.

Example: If the current SQL statement is:

```
select * from activity
```

and you type the following ISQL CHANGE command:

```
change /*/actno/
```

the result in the SQL command buffer is:

```
select actno from activity
```

COLUMN

```
          ┌─1──────────────────────┐
►►──COLumn──┼────────────────────────┼──────────────────────────────────────────────►◄
          └─position_in_select_list─┘
```

COLUMN is an ISQL display command that displays the query result to be formatted so that it begins with the specified column at the left edge of the display.

blank
: causes column 1, or the first displayable column if column 1 is being excluded, to be displayed at the left edge of the display.

position_in_select_list
: is the number of the specified column.

  The number refers to the columns provided by the SELECT statement, not those currently displayed. You must choose a number that represents the desired column's position in the SELECT clause of the SELECT statement that provided the query result.

  If the value of *position_in_select_list* represents a column being excluded, the display starts at the next column to the right that is not being excluded. If there are no more displayable columns to the right, a blank display is the result.

  Specifying a value greater than the number of columns in the query result displays the last column or a blank display if the last column is being excluded.

COUNTER

```
►►──COUNTER──┬─*──────────┬──────────────────────────────────────────────────►◄
             │ ┌────────┐ │
             └─▼─name───┴─┘
```

COUNTER is an operator command that is used primarily to monitor system activity. It is not allowed during a logical unit of work. Before typing COUNTER, you must end any logical unit of work that is in progress.

The SHOW command can only be used when the target application server is a local application server. It cannot be used when the application server is a remote application server.

***** specifies that all counters are to be displayed.

name
is the name of the counters to be specified. Valid names are:

```
BEGINLUW    DBSSCALL    LDIRBUFF    PAGEREAD
CHKPOINT    DEADLCK     LOCKLMT     PAGWRITE
DASDIO      DIRREAD     LOGREAD     RDSCALL
DASDREAD    DIRWRITE    LOGWRITE    ROLLBACK
DASDWRIT    ESCALATE    LPAGBUFF    WAITLOCK
```

For an explanation of these counters, see the *DB2 Server for VSE & VM Operation* manual.

The COUNTER command results in one or more displays. To proceed to the next display, press CLEAR. It is not possible to move backward. You must first type the END command, and then retype the COUNTER command. To exit from the COUNTER command, type END. The COUNTER command is not available on a non-DB2 Server for VSE & VM application server or if you are using DRDA protocol.

Note: If the national language of the application server differs from the national language that the user set for the ISQL session, the messages generated by this operator command are issued in the national language of the application server.

DISPLAY

```
►►──DISPLAY──────────────────────────────────────────────────────────►◄
```

DISPLAY is an ISQL display command that can *only be used in a routine*. When encountered, DISPLAY causes its associated SELECT statement results to be displayed at your display terminal.

Any ISQL display commands placed in the routine after the associated SELECT statement and before the DISPLAY command affect the display at the terminal. You can, in addition, format the display or print the query result by typing ISQL display commands from the keyboard after the query result is displayed, or from the routine after the display is ended and before the END statement that is associated with the SELECT statement whose results are being displayed. To end the display of the query result initiated by the DISPLAY command and to return to the routine, type the ISQL END command from the keyboard.

Example: In the following routine, you select all the rows from the DEPARTMENT table, specify the format commands to be performed on the rows, and display the formatted results on the display terminal. You must type `end` to end the display.

```
select * from department
format separator ' | '
format ttitle 'departments'
display
end
```

To get the same result as the above routine, you can type these lines in a routine:

```
select * from department
display
end
```

to see the DEPARTMENT table. Next, type these commands from the keyboard:

```
format separator ' | '
format ttitle 'departments'
```

and you see the DEPARTMENT table with column separators and a title just like the first example.

END

```
►►──END──────────────────────────────────────────────────────────────►◄
```

END is an ISQL command that ends the display of a query result, an operator command, a SELECT statement in a routine, or an ISQL INPUT command. Query results, operator commands, or the INPUT command can also be ended by pressing the PF3 key (or PF15 key). Because you can reset your PF keys, you can assign the END function to any PF key. The default function key is PF3 (or PF15).

ERASE

```
             ┌──────────────────────┐
             ▼                      │
►►──ERASE────stored_statement_name──┴──────────────────────────►◄
```

ERASE is an ISQL command that erases one or more stored SQL statements.

stored_statement_name
is the name of the stored SQL statement.

EXIT

```
►►──EXIT──────────────────────────────────────────────────────────────►◄
```

EXIT is an ISQL command that ends the current ISQL session. If AUTOCOMMIT is on, the session ends immediately. If AUTOCOMMIT is off, you are prompted for a response so that the current logical unit of work can be committed or rolled back before you exit the session.

FORMAT

```
                ┌──────────────────────────────────────────────────────────────┐
                ▼                                                              │
►►──FORMAT──────┬─┬─BTITLE─┬──┬─────────┬──────────────────────────────────────┬┴──────────────►◄
                │ └─TTITLE─┘  ├─string──┤                                      │
                │             └─ERASE───┘                                      │
                ├─COLumn──column_id──┬─DPLACes──integer─┬──────────────────────┤
                │                    ├─NAME──string─────┤                      │
                │                    ├─WIDTH──integer───┤                      │
                │                    │         ┌─OFF─┐  │                      │
                │                    └─ZEROs───┼─────┼──┘                      │
                │                              └─ON──┘                         │
                ├─EXCLude──┬─────────┬──┬─column_id───────────────────┬────────┤
                │          └─ALL BUT─┘  │     ┌───────────┐           │        │
                │                       └─(───▼─column_id─┴───)───────┘        │
                ├─┬─GROUP────┬──┬─┬────────┬──┬─column_id──────────────┬─┬─────┤
                │ ├─SUBTotal─┤  │ └─EXCEPT─┘  │     ┌───────────┐      │ │     │
                │ └─TOTal────┘  │             └─(───▼─column_id─┴───)──┘ │     │
                │               └─ERASE──────────────────────────────────┘     │
                ├─INCLude──┬───────────────────────────────────────┬───────────┤
                │          └─┬──────┬──┬─column_id──────────┬──────┘           │
                │            └─ONLY─┘  │     ┌───────────┐  │                  │
                │                      └─(───▼─column_id─┴──)┘                 │
                ├─NULL──string─────────────────────────────────────────────────┤
                │ ┌─ON──┐                                                      │
                ├─┼─────┼──────────────────────────────────────────────────────┤
                │ └─OFF─┘                                                      │
                │          ┌─ON──┐                                             │
                ├─OUTLINE──┼─────┼─────────────────────────────────────────────┤
                │          └─OFF─┘                                             │
                │                 ┌─2─────────┐                                │
                ├─SEParator────┬──┼───────────┼──BLANKs─┬──────────────────────┤
                │              │  └─integer───┘         │                      │
                │              └─string─────────────────┘                      │
                │          ┌─20────────┐                                       │
                └─VARChar──┼───────────┼───────────────────────────────────────┘
                           └─integer───┘
```

FORMAT is an ISQL display command that controls the format of the query result currently being displayed. Modifications made to the display format by a FORMAT command are reflected on the current display and the printed output obtained from any subsequent PRINT command.

You can perform formatting only on the first 45 columns of a query result.

Any formatting command containing the keywords EXCLUDE, GROUP, INCLUDE, ON/OFF, OUTLINE, SUBTOTAL, or TOTAL, causes the query to be reexecuted.

Note: You can use DBCS characters in strings or column lists.

BTITle
: specifies the bottom title to be printed on reports. This bottom title is centered on the bottom line of the report. Unless specified, no bottom title is printed.

  string
  : is the characters to use for the bottom title on a subsequent PRINT command.

The maximum length of the bottom title is that which fits on a line of the page (up to a maximum of 100 characters). If the bottom title exceeds the maximum, only those characters that fit on the line are printed.

Enclose the title in *single* quotation marks if it contains any blanks, but do not include any single quotation marks in the title itself.

ERASE
causes the current bottom title to be deleted, resulting in no bottom title.

If FORMAT BTITLE is issued without either ERASE or a character string, the current bottom title is displayed if it does not contain DBCS data. If the title *does* contain DBCS data, message `ARI7970I` is issued.

TTITle
specifies the top title to be printed on reports. This top title is centered on the top line of the report between the date and page number. If no top title is specified, the first 100 characters of the SELECT statement are used for the top title on printed reports.

string
is the characters to use for the top title on a subsequent PRINT command.

The maximum length of the top title is that which fits on the top line between the date and page number (to a maximum of 100 characters). If the top title exceeds the maximum, only those characters that fit between the date and page number are printed.

Enclose the title in *single* quotation marks if it contains any blanks, but do not include any single quotation marks in the title itself.

ERASE
causes the current top title to be deleted and the default to be used. The default TTITLE is the first 100 characters of the SELECT statement.

If FORMAT TTITLE is issued without ERASE or a character string, the current top title is displayed if it does not contain DBCS data. If the title *does* contain DBCS data, message `ARI7970I` is issued.

COLumn
provides display formatting for a particular column.

column_id
specifies the column to be formatted.

If a number *n* is specified, it identifies the column to be formatted as the nth column of the query result. The number refers to the position of the columns provided by the SELECT statement, not those currently displayed. Therefore, you must choose a number that represents the position of the desired column in the SELECT clause of the SELECT statement that provided the query result.

If a number is not specified, *column_id* refers to the current column heading of the column to be formatted. Enclose the name in single quotation marks if it either contains a blank or refers to a column with a solely numeric heading.

If the *column_id* represents a column being excluded, the formatting specified is performed on the excluded column although you cannot see the formatting until the column is included.

The following keywords describe how the current display is to be formatted for the column specified by *column_id*:

DPLACEs
specifies the number of decimal places (*integer*) to be displayed for a numeric field. Rounding is *not* performed. The DPLACEs must be less than the column width.

NAME
specifies a column heading to be used for the display.

string
is the actual column heading to be displayed.

A maximum of 30 characters can be used for the column heading. All characters except single quotation marks are valid. Enclose the column heading in single quotation marks if it contains a blank, but do not include any single quotation marks in the column heading itself.

WIDth
specifies the display length attribute (*integer*) of the column.

integer
For character type columns, the leftmost *integer* characters are displayed.

For numeric type columns, the leftmost *integer* significant digits counting the sign and decimal marker, if any, are displayed. The sign character for a positive number is a blank.

For VARCHAR type columns, characters up to the current setting of VARCHAR (using a SET or FORMAT command) are displayed. For GRAPHIC type columns, integer represents the number of DBCS characters. Two bytes are reserved for the SO/SI characters.

ZEROs
specifies whether leading zeros for numeric columns are displayed (ON) or not (OFF).

EXCLude
specifies columns to be excluded (omitted) from the display. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

ALL BUT
indicates that all columns except those specified are excluded. For example:

```
format exclude all but (1 job 4)
```

includes the JOB column, and columns 1 and 4, but excludes all other columns.

column_id
specifies the column to be formatted.

If a number is specified, it identifies the column to be formatted as the nth column of the query result. The number refers to the position of the columns provided by the SELECT statement, not those currently being displayed. Therefore, you must choose a number that represents the position of the desired column in the SELECT clause of the SELECT statement that provided the query result.

If a number is not specified, *column_id* refers to the current column heading of the column to be formatted. Enclose the name in single quotation marks if it contains a blank or refers to a column with a solely numeric heading.

When more than one column is specified, separate the column_ids with a blank and enclose them in parentheses.

For example, the command:

```
format exclude (3 5)
```

causes the third and fifth column of the original position in the query result to be excluded from the display. The command:

```
format exclude job
```

prevents the JOB column from being displayed during the current query result. If JOB is selected two or more times, only the first occurrence of the JOB column is excluded.

GROUP
specifies the columns to outline (when outlining is on) and the columns to use for determining when subtotals are taken. Subtotals are taken whenever the values change in the columns specified. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

SUBTotal
specifies the columns in which subtotals are to be calculated. Subtotals are taken whenever the values change in the columns being grouped. A (final) total is also provided for the columns unless otherwise specified by a FORMAT TOTAL command. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

TOTal
specifies the columns in which (final) totals are to be calculated. If not specified, totals are provided for all columns being subtotaled. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

EXCEPT
specifies the columns to be grouped or totalled or subtotaled except those specified by *column_id*.

column_id
is the name or position of each column in the query result to be grouped for outlining, subtotals, or totals.

When used with the SUBTOTAL and TOTAL keywords, column_id specifies the columns on which subtotals or totals are to be calculated.

If a number is specified, it identifies the column to be formatted as the nth column of the query result. The number refers to the position of the columns provided by the SELECT statement, not those currently displayed. Therefore, you must choose a number that represents the position of the desired column in the SELECT clause of the SELECT statement that provided the query result.

If a number is not specified, *column_id* refers to the current column heading of the column to be formatted. Enclose the name in single quotation marks if it contains a blank or refers to a column with a solely numeric heading.

When specifying more than one column, separate the column_ids with a blank and enclose them in parentheses.

If the column represents one that you are excluding, ISQL groups the columns using that excluded column, although you do not see the excluded column.

Subtotals or totals are calculated on the excluded column. You cannot see the subtotals or totals until the column is included. When arithmetic errors occur, the value of the column in error is calculated as zero.

The usual use of GROUP is to first order the rows of the columns that you want grouped. Use an ORDER BY clause in the SELECT statement issued to obtain the query result. The left to right ordering of the columns themselves depends on the order in which they appear in the SELECT clause of the SELECT statement.

Rows containing arithmetic errors from an outer select are displayed together at the end of the list, followed by rows containing NULL values. Rows containing arithmetic errors are displayed as asterisks (*).

ERASE
deletes all the previous specifications of GROUP for the current query result. When used with SUBTOTAL, ERASE suspends subtotals. When used with TOTAL, ERASE suspends totals.

INCLude
reverses the effect of a previous FORMAT EXCLUDE command. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

ONLY
indicates that only those columns specified are displayed; all others are excluded.

Note: Do not use ONLY for a column name with an INCLUDE keyword. In this case, use its column number.

column_id
specifies the column to be formatted.

If a number *n* is specified, it identifies the column to be formatted as the nth column of the query result. The number refers to the position of the columns provided by the SELECT statement, not those currently being displayed. You must choose a number that represents the position of the desired column in the SELECT clause of the SELECT statement that provided the query result.

If a number is not specified, *column_id* refers to the current column heading of the column to be formatted. Enclose the name in single quotation marks if it contains a blank, or if it refers to a column with a solely numeric heading.

When more than one column is specified, separate the column IDs with a blank and enclose them in parentheses.

When arithmetic errors occur, the value of the column in error is calculated as zero.

The columns that are not mentioned in the INCLUDE command, and that are not currently being excluded, continue to participate in the display. If INCLUDE is issued by itself and with no options, all excluded columns are restored.

NULL
specifies the characters to display for null fields.

string
specifies the actual characters to display (up to a maximum of 20). If blanks are included, you must enclose the string in single quotation marks, but do not use single quotation marks in the string itself.

For example, the following command,

```
format null empty
```

causes the word EMPTY to be displayed for all null fields.

A question mark (?) is displayed for null values unless otherwise specified with a FORMAT or SET command.

ON
OFF
controls the status of outlining, subtotals, and totals on query results. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result. Until OFF is specified, outlining, subtotals, and totals are active.

ON
permits outlining, subtotals, and totals. The ON status stays in effect for the current query result until you type FORMAT OFF.

OFF
suspends outlining, subtotals, and totals.

OUTLINE
controls the outline report format for columns specified with FORMAT GROUP. When SQL processes a FORMAT command containing this keyword, it reexecutes the query and repositions the display to the top of the query result.

If OUTLINE is not specified, outlining is performed whenever GROUP is specified unless you type FORMAT OFF.

ON
specifies that successive duplicate values in grouped columns are repeated only when they are the first line at the top of the display or at the beginning of each page on printed reports. If the first line is a subtotal line or a blank line between groups, successive duplicate values are not displayed or printed at the beginning of the next group.

OFF
specifies that successive duplicate values in grouped columns are to be displayed wherever they occur.

SEParator *integer* **BLANKs**
specifies the number of spaces (*integer*) to be displayed between columns. The maximum number of blanks that can be specified is 254. Unless otherwise specified with a FORMAT or SET command, the separation between columns consists of two blanks.

SEParator *string*
specifies the characters to be displayed between columns. If blanks are included, the string must be enclosed in single quotation marks, but do not include any single quotation marks in the separator itself.

For example, if you want a vertical line between the columns, you type:

```
format separator ' | '
```

which places a blank, a vertical bar, and a blank between all columns. The maximum number of characters that can be used for a separator is 254.

VARChar
specifies the display width of variable length columns.

integer
is the length desired up to 254. Unless otherwise specified with a SET command, ISQL displays only the first 20 characters of a variable-length column.

The SET command value for VARCHAR columns can be overridden for a particular query by specifying the desired value with this keyword on the FORMAT command.

When you type a FORMAT VARCHAR, the SELECT statement is reissued and you are returned to the beginning of the query result.

Example: The following FORMAT commands used during a query exclude the first column, change the name of the PRSTAFF column heading to ESTMEAN, display all projected mean staff numbers in this column with three decimal places, and display leading zeros:

```
format exclude 1
format column prstaff name estmean
format column estmean dplaces 3
format column estmean zeros on
```

Note: If you rename a column heading, further FORMAT commands that refer to that column by name must use the new name. When referring to the column by a number, you must specify its original position in the SELECT clause of the SELECT statement.

You can specify more than one keyword in a FORMAT command. For example, the FORMAT commands described above can be expressed with a single command:

```
format exclude 1 column prstaff name estmean dplaces 3 zeros on
```

By using more than one keyword in a single FORMAT command, you can reduce the amount of data you must type and improve the performance of ISQL.

FORWARD

```
►►──FORWARD──┬───────────────┬──────────────────────────────────────────────────►◄
             ├─rows_integer──┤
             └─MAX───────────┘
```

FORWARD is an ISQL display command that lets you move your display forward through a query result.

rows_integer
is the number of rows you want to move forward.

MAX
causes the last display of the query result to be displayed along with an indication of the number of rows in the query result and the Query Cost Estimate (QCE) message. For query results that contain many rows, a FORWARD MAX command can take a long time.

If nothing is specified, the display moves forward one-half of a display.

Scroll forward through the query result one-half of a display at a time by typing FORWARD. (You can also press PF8 or PF20.) Because you can reset your PF keys, you can assign the FORWARD function to another PF key.

You can activate FORWARD by pressing ENTER if it is the first display command to be issued for a query result. The display moves forward one *entire* display each time you press ENTER.

HELP

```
►►──HELP──┬──────────────┬──────────────────────────────────────────────────────────►◄
          ├─CONTENTs─────┤
          └─topic_name───┘
```

This ISQL command retrieves online HELP information. This online HELP information is for ISQL users who need quick *reference* information at their display for:

SQL statements
ISQL commands
Messages, codes, and SQLSTATEs.

Note: The online HELP information is not serviced by the IBM Support Center. The information is extracted from this book, and from the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals. Use the readers' comment form in the back of these books to express concerns and comments on this information.

You can view this information in the language of your choice if your site chose to install different language versions of the online HELP information. See the SET command later in this chapter for more information on setting the language of your choice.

Online HELP information is stored as a table. After you retrieve a topic, you can use any of the ISQL display commands, or PF keys that provide display commands, to assist in viewing the text. You can also type SQL statements or ISQL commands at this time. You can format retrieved topics with the ISQL FORMAT command and print them with the ISQL PRINT command. A top title is provided for printed topics.

CONTENTs
displays the available online topics and the correct names to use in retrieving specific topics.

topic_name
is the name of the topic to be retrieved. It can be one or more words and can be placed within quotation marks. Most *topic_names* are either a statement name (such as *SELECT* or *INSERT*), a message number (such as *ARI7399I* or *ARI7307A*), a message code (such as *-205* or *100*), or an SQLSTATE (such as *SQLSTATE 01512*). For example, to retrieve online HELP information for the UPDATE statement, type:

```
help update
```

If the HELP command is issued with no parameters, a description of how to use the HELP command is returned along with a list of the available topics. The HELP command without additional parameters can be invoked by pressing PF1 (or PF13). Because you can reset your PF keys, you can assign the HELP function to any PF key. The default function key is PF1 (or PF13).

You can type SQL statements while the online HELP information is displayed.

HOLD

```
►►──HOLD──sql_statement──────────────────────────────────────────────►◄
```

HOLD is an ISQL command that prevents an SQL statement from being processed when it is typed. The SQL statement is placed in the SQL command buffer and remains there until it is replaced with another SQL statement. You can check the SQL statement for typing errors before it is processed by a START command. You can also type an SQL statement containing placeholders and substitute values for the placeholders when the statement is started using a START command.

sql_statement
is the SQL statement to be held. (ISQL commands cannot be held).

The following example illustrates the use of a HOLD command to place an SQL statement in the command buffer without executing it:

```
hold select * from employee
```

The HOLD command can also be invoked by pressing PF9 (or PF21) prior to, during, or after typing. By pressing PF9 instead of ENTER, the SQL statement typed is placed in the SQL command buffer without being processed. The HOLD PF key does not store the command in the SQL command buffer; you must press ENTER after pressing HOLD PF. The default key for the HOLD function is PF9 (or PF21).

IGNORE

```
►►──IGNORE──────────────────────────────────────────────────────────►◄
```

IGNORE is an ISQL command that nullifies a command, statement, or data that is being typed on multiple input lines. Type it at the start of the input area.

The following example shows how the IGNORE command can be used to correct a table-naming error:

```
select projno,actno,acstaff -
from emp_act -
ignore
select projno,actno,acstaff -
from proj_act -
where projno = 'MA2100'
```

INPUT

```
►►──INPut──┬─table_name─┬──┬──────────────────────────────┬──────────────────────────────►◄
           └─view_name──┘  │    ┌─,─────────────┐          │
                           └─(──▼─column_name──┴─)─────────┘
```

INPUT is an ISQL command that enables you to insert several rows of data into a table or view.

INPUT is issued to identify the table, view, or column(s) where the data is inserted.

table_name
view_name
: is the name of the table or view in which data is to be inserted. A *view_name* can only be used if it applies to a single table. For more information, see “Updating Tables on Which Views are Defined” on page 162. The names can contain DBCS characters.

column_name
: is one or more column names, separated by commas, into which data is to be inserted. The order in which the column names are specified determines the order in which the data must be typed. All columns of the new row that are not listed receive a null value, and unlisted columns must have been defined to accept null values or an error occurs. This is because the INPUT command is essentially inserting a new row of data, and null values are inserted into any columns not specified. Omitting the column_name is the same as naming all the columns of the table in their created order. The names can contain DBCS characters.

Successful execution of the INPUT command causes the column names and their data types to be displayed in the order in which the data must be typed. Data can then be typed one row at a time. Press the ENTER key after each row is typed.

When typing data:

- Use commas to separate each data item of a row.
- Enclose the data item in single quotation marks if it is a CHAR, VARCHAR, DATE, TIME, or TIMESTAMP data type.
- Do not enclose the keyword **NULL**, or any of the special registers, such as CURRENT SERVER, in single quotation marks.
- If your data includes a single quotation mark, type two single quotation marks. When you type:

  'julie's book shop'

  JULIE'S BOOK SHOP is inserted into the table.
- If you are typing graphic data, you type:

  G'so...si'

  where so stands for shift-out character, si stands for shift-in character, and **...** is graphic data (a DBCS character string).

Note: You can use N' as a synonym for G'

- If you are typing hexadecimal data, you type X'F140F2'.
- If all the data for a single row does not fit in the input area, type the continuation character. Press ENTER to continue.
- Be sure to include a space before the continuation character, if required, because the continuation character does *not* provide one.
- When null values are allowed, you can type NULL to insert a null value for the data-item value.

After you press ENTER for a row of data, the data is moved to the output area of the display and the input area is cleared so that another row of data can be typed. When all the data has been typed, type the END command to signify the end of input data.

With AUTOCOMMIT on, the data you type is not committed to the table until the INPUT command is ended by the END command or the END PF key (usually PF3). You can use the ISQL SAVE command to commit data in the table prior to typing an END command.

The SAVE command stores all data typed since the previous SAVE command or, if one had not been typed, since the start of the INPUT command. It has the same effect as the END command but lets you continue to type data.

In addition to storing data prior to ending an INPUT command, you can also prevent the storing of data typed since the last SAVE command or since the start of the INPUT command if no SAVE has been issued. You type the ISQL BACKOUT command instead of more data.

If AUTOCOMMIT is off, the data you type on an INPUT command is not committed to the table until you type a COMMIT command after ending the INPUT command. With AUTOCOMMIT off, the SAVE and BACKOUT commands have no effect. See “Understanding Referential Integrity” on page 72 for more information about maintaining the integrity of data in tables.

Example: The following illustrates how you can fill the ACTIVITY table by using the INPUT command:

```
input activity
  .
  .
  .
160,'ADMDB','Adm databases'
  .
  .
  .
100,'TEACH','Teach classes'
end
```

Interactive Select

interactive-select-statement

```
>>─┤ fullselect ├─┬──────────────────────────────────────────────────┬─┬──────────────────┬─><
                  │                ┌─,──────────────────────────┐   │ └─WITH──┬─RR─┬──┘
                  │                ▼                    ┌─ASC──┐ │   │         ├─CS─┤
                  └─ORDER BY─────┬─column_name─┬──┼──────┼─┴───┘         └─UR─┘
                                 └─integer─────┘  └─DESC─┘
```

fullselect:

```
                              ┌──────────────────────────────────────────┐
                              ▼                                          │
├──┬─subselect────┬──────────┬──────────────────────────────────────────┬┴─────────────────────┤
   └─(fullselect)─┘          └─┬─UNION─────┬───┬─subselect────┬──────────┘
                               └─UNION ALL─┘   └─(fullselect)─┘
```

subselect:

```
            ┌─ALL──────┐
├──SELECT───┼──────────┼──┬─*───────────────────────────┬──────────────────────────>
            └─DISTINCT─┘  │  ┌─,──────────────────────┐ │
                          │  ▼                        │ │
                          └──┬─expression─────────────┴─┘
                             ├─table_name.*─────────┤
                             ├─view_name.*──────────┤
                             └─correlation_name.*───┘

      ┌─,─────────────────────────────────────────┐
      ▼                                           │
>──FROM──┬─table_name─┬──┬──────────────────────┬──┴─┬──────────────────────────────┬──────>
         └─view_name──┘  └─correlation_name─────┘    └─WHERE──search_condition───┘

>──┬─────────────────────────────────┬──┬───────────────────────────────┬──────────────────┤
   │              ┌─,─────────────┐  │  └─HAVING──search_condition───┘
   └─GROUP BY─────▼─column_name───┴──┘
```

The interactive-select statement retrieves data from a table or view. The data retrieved can only be viewed; you cannot change it. For more information about the terminology used in this diagram, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

If a SELECT statement contains a placeholder in the WHERE, GROUP BY, ORDER BY, or HAVING clauses, FORMAT information is saved until the next SELECT statement is entered. Formatting information is saved permanently if the SELECT statement is stored. However, if the SELECT or FROM clauses contain placeholders or parameters, FORMAT information is not saved. In addition, FORMAT information is not saved if you change any data in the SELECT or FROM clauses by using the CHANGE command.

When you enter a SELECT statement from a terminal, the row length of the information you can see at one time is limited by the terminal screen size.

The number of bytes per row includes the bytes used as column separators.

UNION
: combines two or more queries into a single query by merging the rows returned by each query. Duplicate rows are eliminated.

  ALL
  : combines the results of two or more queries without eliminating the duplicate rows.

ALL
specifies that duplicate values are to be selected. If ALL is specified when you select all department numbers from the PROJECT table, each and every department number listed in the department number column is selected. This is the default.

DISTINCT
specifies that duplicate values are not to be selected. DISTINCT can be used only once for each SELECT statement. You can use it to eliminate duplicates from the SELECT result:

```
select distinct deptno from project
```

You can also use it to eliminate duplicates from a column function:

```
select count (distinct deptno) from project
```

If you specify DISTINCT, the maximum number of columns you can put in the *expression* list is 16. In addition, the length of the encoded key derived from the *expression* list must not exceed 255 characters. That is, the sum of the lengths of the columns in the *expression* list, plus approximately 25% of the lengths of those columns that are of varying-length character type, must not exceed 255 EBCDIC or 127 DBCS characters.

***** indicates that the data in all the columns is to be selected.

Note: If the table contains more than 45 columns, only the first 45 columns will be displayed. To see the other columns in the table, you can create a view. A view may contain no more than 140 columns. If more columns are needed, you can create additional views.

expression
is a definition of the data desired. An *expression* may be a column name, a constant, a character expression, a special register, a column function, a scalar function, an arithmetic expression, or a labeled duration.

You can specify a list of expressions, separating each with a comma, and the items are retrieved in the same left-to-right order as they appear in the list. The value of USER is interpreted as a CHAR(8) string whose value is the user ID of the user currently connected. The maximum number of columns that you can specify in the list is 45.

You can use the following operators to connect numeric data types:

+	(plus, add)
–	(minus, subtract)
*	(times, multiply)
/	(divided by)

If you use these operators for numeric data types, see the *DB2 Server for VSE & VM SQL Reference* manual for information about data conversion. See “Selecting Information Using Datetime Arithmetic” on page 42 for information on arithmetic operations for date and time values.

You can use the concatenation operator, CONCAT, to join two or more compatible operands to form a string. An operand may be a column, a name, a constant, or an expression.

An operand can be the result of an expression. For example, if the USER special register is used, it is treated as CHAR(8). If the CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP, or CURRENT TIMEZONE special registers are used, the values are treated as the character representation of the value in the format defined by the system. A datetime value can be concatenated with a character string because the datetime data types are compatible with character data types.

When varying-length operands are concatenated, only the actual length of the operand is concatenated.

The result of the expression is the concatenation of the operand expressions. The resulting data type is null if either operand is nullable. The resulting data type is character if both operands are CHARACTER. If both operands are GRAPHIC, the result data type is GRAPHIC. If both operands are fixed length, the resulting data type is fixed length. If either operand is varying length, the resulting data type is a varying length string. The associated defined length is the sum of the defined lengths not exceeding 254 bytes.

In the following example, the employee number and the first name of the employee are concatenated with a hyphen between them:

```
select empno concat '-' concat
firstnme from employee
```

If a column function is used in an *expression*, all expressions in the list must contain a column function unless grouping is being performed. An example is:

```
select min(edlevel),avg(bonus) from employee
```

To use constants in the list of expressions of union operations, see "Combining Multiple Queries" on page 151.

*table_name.**
*view_name.**
*correlation_name.**
identifies the table or view to which the column belongs. The asterisk (*) can be replaced with a column name. These prefixes are especially useful for differentiating columns that have the same name, but that belong to different tables or views. The *correlation_name* can be used to simplify a query, or to join a table to itself. See "*correlation_name*" below for more information.

FROM *table_name*
FROM *view_name*
identifies the table or view from which data is to be selected.

table_name
view_name
is the name of the table or view.

You can further qualify the table or view by specifying the owner of the table or view. You must separate the owner's name from the table name or view name with a period. The owner's name is unnecessary for tables or views that you own. You must have the SELECT privilege or DBA authority to select information from tables or views owned by other users.

correlation_name
is the name you define as an alternative name for the table or view to be selected. It can be any string up to 18 characters long, and must begin with a letter.

WHERE *search_condition*
is one or more conditions to apply in selecting data.

GROUP BY *column_name*

A query can have the column functions SUM, AVG, MAX, MIN, and COUNT applied to groups of rows that have matching values in a column. Rows can also be grouped by matching values in more than one column. The definition of the groups is specified with a GROUP BY clause.

column_name

is the name of one or more columns, separated by commas, to be used when forming a group.

For example, the maximum, minimum, and average activity staff *for each project* in the PROJ_ACT table could be selected by the following query:

```
select projno,max(acstaff),min(acstaff),avg(acstaff) -
from proj_act -
group by projno
```

When a query uses the grouping feature, it returns only one result row for each group. Therefore, the items selected by such a query must be properties of the groups, not properties of individual rows. The *expression* list may contain columns that are also in the GROUP BY clause together with column functions on any columns. It may not contain any non-grouped column without a column function. If the column function COUNT(*) is used, it evaluates to the number of rows in the group.

If any rows have a null value in a grouped column, ISQL groups the null values in those columns together. The null values returned may be due either to unknown column values or to arithmetic exception errors.

A grouping query may have a standard WHERE clause that serves as a filter, keeping only those rows which satisfy the *search_condition*. The WHERE clause filters out the non-qualifying rows before the groups are formed and the column functions are computed.

The following example query finds the average and minimum activity staff for each project, considering only activities whose starting date is 1 January 1982:

```
select projno,avg(acstaff),min(acstaff) -
from proj_act -
where acstdate = '1982-01-01' -
group by projno
```

HAVING *search_condition*

is one or more conditions that apply to groups. ISQL returns a result only for those groups that satisfy the condition. The HAVING clause may contain one or more group-qualifying conditions connected by ANDs and ORs. Each group-qualifying condition compares some property of the group, such as AVG(ACSTAFF), with another group property or with a constant.

The following example query lists the maximum and minimum activity staff for various projects in the PROJ_ACT table, considering only projects that have more than three activities:

```
select projno,max(acstaff),min(acstaff) -
from proj_act -
group by projno -
having count(*) > 3
```

One of the functions in a HAVING clause may specify DISTINCT (for example, COUNT(DISTINCT PROJNO)). However, DISTINCT may be used only once in a query. It may not be used in both the *expression* list and the HAVING clause.

It is possible, though unusual, for a query to contain a HAVING clause but no GROUP BY clause. In this case, the entire table is treated as one group. If the HAVING condition is true for the table as a whole, the selected result, which must consist entirely of column functions, is returned.

ORDER BY
orders, or sorts, the rows to be retrieved by the column(s) specified. A maximum of 16 columns may be specified in the ORDER BY clause.

column_name
refers to a column name in the *expression* list. For example, to order a query primarily by the values in the ACTNO column and secondarily by the values in the PROJNO column, you would type:

```
select actno,projno,acstdate,acendate -
from proj_act -
order by actno,projno
```

integer
refers to the items in the *expression* list. For example, to order a query primarily by the first item and secondarily by the third item of the list, you would type:

```
select projno,acstdate,acendate,acstaff -
from proj_act -
order by 1,3
```

These items may be columns or more complex expressions such as BONUS+COMM.

ASC
DESC
indicates the order in which results are returned: either ascending (ASC) or descending (DESC). The default is ASC. For example, to indicate ascending order on item 3 and descending order on item 5, you could type:

```
order by 3,5 desc
```

because ASC is the default.

In contrast, to indicate descending order on item 3 and item 5, you would type:

```
order by 3 desc,5 desc
```

Blanks sort first in ascending order, last in descending order, and are ignored if the data type is VARCHAR or VARGRAPHIC.

ISQLTRACE

```
►►──ISQLTrace──┬─DUMP────┬──────────────────────────────────────────────►◄
               ├─DISPLAY─┤
               └─integer─┘
```

The ISQLTRACE command traces activity within ISQL. This ISQL command traces calls to and returns from other ISQL modules, SQL return codes, and ISQL messages. Trace information is saved in storage for dump debugging.

DUMP
specifies that the trace information is to be printed. ISQL creates an unformatted storage dump hardcopy of the trace table.

DB2 Server for VM

ISQL uses the CP DUMP command to dump to the lowest priority virtual printer defined.

DB2 Server for VSE

ISQL issues a CICS dump to the CICS dump data set.

You must run a job to print the dump data set.

DISPLAY
specifies that the formatted trace table is to be displayed on the display. If more than 50 entries or more than the specified *integer* number of entries have been made in the trace table, the entries wrap. The more recent entries are written over the earlier entries. As a result, only the last 50 entries or *integer* number of entries in the table are displayed. The entries are displayed in reverse order of the order that they were put into the trace table.

integer
changes the size of the trace table. Replace *integer* with the number of trace entries that are to be contained in the trace table. *integer* must be a number from 50 to 1000.

LEFT

```
            ┌─1───────┐
►►──LEFT────┼─────────┼──────────────────────────────────────►◄
            └─integer─┘
```

LEFT is an ISQL display command that causes the display to start *integer* columns to the left, counting from the leftmost column on the display and as long as no columns are being excluded. The number refers to the columns provided by the SELECT statement, not those currently displayed. You must use the order of the columns specified in the SELECT clause of the SELECT statement that provided the query result to determine the value to specify for integer. If the value of *integer* represents a column being excluded, the display starts at the next displayable column to the right of the excluded column.

Specifying a value greater than the number of columns remaining to the left, starts the display at column 1 or at the first displayable column if column 1 is excluded.

If no number is specified, the display moves one column to the left. No change in the display occurs if that column is excluded.

LIST

```
                   ┌─────────────────────────┐
►►──LIST───┬─SQL───▼─┬─stored-statement-name─┬─┬────────────────────────────────►◄
           │         └─*─────────────────────┘ │
           │       ┌──────────────────┐        │
           └─SET───▼─┬─*──────────────┬────────┘
                     ├─AUTOCommit─────┤
                     ├─CASE───────────┤
                     ├─CLAss─(1)──────┤
                     ├─CONTInue───────┤
                     ├─COPies─(1)─────┤
                     ├─COSTest────────┤
                     ├─DECimal────────┤
                     ├─ISOLation──────┤
                     ├─LANGuage───────┤
                     ├─NULL───────────┤
                     ├─PAGEsize───────┤
                     ├─PRINTRoute─(1)─┤
                     ├─RUNMode────────┤
                     ├─SEParator──────┤
                     └─VARChar────────┘
```

Note:
[1] The parameters CLAss, COPies, and PRINTRoute are not applicable to VM.

LIST is an ISQL command that lists information about stored SQL statements or the settings of operational characteristics set by the SET command.

SQL
lists stored SQL statement(s) on the display. See "STORE" on page 257 for more information.

stored_statement_name
is the name of the stored SQL statement to be listed. The entire statement stored with the specified name is displayed.

*****
specifies that all your stored SQL statements are to be listed. The name of each statement and its first 50 characters are displayed.

SET
lists the current operational characteristics in effect for the specified function of the SET command. For example, if you type:

```
list set continue
```

the current continuation character is displayed. Valid functions are any of those functions performed by the SET command.

If * is specified, the current operational characteristics in effect for all SET command functions are listed. For a description of the SET characteristics, see "SET" on page 245. for more information.

Examples: The LIST command can be used with more than one keyword option in a single command. For example, if you want to display more than one operational characteristic, you can type:

```
list set page case null
```

If you want to display more than one stored SQL statement, you can type:

```
list sql myquery query1
```

Stored SQL statements and operational characteristics may not be listed at the same time.

PRINT

```
            ┌─────────────────────────┐
►►──PRInt───▼─┬───────────────────────┬┴─────────────────────────────────────────────►◄
              │          ┌─?─────────┐ │
              ├─CLAss────┴─character─┴─┤
              │           ┌─1───────┐  │
              ├─COPies────┴─integer─┴──┤
              ├─TERMid──termid─────────┤
              ├─DESTid──wkstat─────────┤
              ├─SYStem─────────────────┤
              └─TOUSER───┬──────────┬──┘
                         └─userid───┘
```

DB2 Server for VSE

PRINT is an ISQL display command that requests printed copies of a query result by sending it to the system printer, POWER remote printer, or CICS/VSE terminal.

DB2 Server for VM

PRINT is an ISQL display command that requests printed copies of a query result by sending it to the system printer.

The data that is printed is based on the query result obtained by the SELECT statement.

All the rows of the query result, as modified by FORMAT commands, are printed regardless of the position of your display of the result. A query result that is longer than what can be viewed on one display, can be printed with one PRINT command. The printed report starts with the column and the character position within the column that is at the left edge of the display when the PRINT command is typed. The number of characters of each row that is printed depends on the setting of the PAGESIZE WIDTH value specified by the SET command. You receive a message if the width of the row of data exceeds the WIDTH setting.

The output class to use can be obtained from the people responsible for data processing at your location.

DB2 Server for VSE

Note: CLASS and COPIES cannot be specified for a CICS/VSE terminal printer.

DB2 Server for VM

Note: You must use the CP TAG and CP SPOOL commands to direct printed output.

CLAss
specifies the output class.

character
is the output class desired. For DB2 Server for VSE, the output class can be a letter (A to Z).

For DB2 Server for VM, the CLASS value can be an integer (0 to 9) or a letter (A to Z). The default is class A.

? (DB2 Server for VSE only)
specifies that the default printer class of the system is to be used.

For DB2 Server for VSE, if CLASS is not included in the PRINT command at all, the current class set by the SET command is used. If no class has been set by a SET command, the default printer class for the system is used.

For DB2 Server for VM, there is no SET command for CLASS. You can set the CLASS value by a CP SPOOL command.

COPies
specifies the number of copies.

integer
is the number of copies desired. For DB2 Server for VSE, you can request up to 99 copies. For DB2 Server for VM, you can specify a COPIES value from 1 to 255.

If this keyword is not specified, one copy is the default unless otherwise determined by a SET command (in DB2 Server for VSE).

DB2 Server for VM

You can also use the CP SPOOL command to specify the number of copies. The value for copies that is specified on the CP SPOOL command remains in effect until another value is specified by another CP SPOOL command.

Note: If you specify the number of copies on the PRINT command after you specified it using a CP SPOOL command, the PRINT command quantity is used for that print operation. All following PRINT commands use the quantity specified by the CP SPOOL command, unless you specify it using the **COPIES** keyword.

Because you can reset your PF keys, you can assign the PRINT function to any PF key. The default function key is PF4 or PF16.

TERMid (DB2 Server for VSE only)
specifies that printed output is to be directed to the designated CICS/VSE terminal.

termid
is the terminal identifier of the CICS/VSE terminal that you want. *termid* must be from one to four alphanumeric characters.

DESTid (DB2 Server for VSE only)
specifies that the printed output is to be directed to the designated POWER remote workstation.

wkstat
is the ID of the desired remote workstation. *wkstat* can be any number from 0 to 250. When *wkstat* is 0, the printed output is to be directed to the system printer.

SYStem (DB2 Server for VSE only)
specifies that the printed output is to be directed to the system printer.

TOUSER (DB2 Server for VSE only)
specifies that the printed output is to be directed to a user identified by *userid*.

userid
is the VSE POWER user identifier of the user to whom the output is being spooled. An identifier cannot be longer than 8 alphanumeric characters.

If you enter the PRINT TOUSER command without specifying a *userid*, ISQL spools the output to the ISQL user ID used at time of log on.

The PRINT TOUSER *nnn* command has the same effect as the PRINT DESTid *nnn* command, where *nnn* is the ID of the remote workstation to which you want to direct your printed output. This ID can be any number from 1 to 250.

DB2 Server for VSE

Any TERM, DEST, or SYS indication on the PRINT command applies for that particular PRINT operation only.

The PRINT command can be started by pressing PF4.

Each printed page is numbered and dated at the top. A title is automatically provided at the top of each printed page. This title consists of the first 100 characters of the SELECT statement issued unless you specified your own title with an ISQL FORMAT command.

You can specify more than one keyword option on a single PRINT command. For example, the following command specifies both the class and number of copies:

```
print copies 3 class a
```

DB2 Server for VM

If there is an error in the command, all valid changes are made until the mistake is determined. For example, in the following command, you type a $ sign instead of the letter a:

```
print copies 3 class $
```

The command sets the number of print copies to 3. However, the class remains at the default class A, because $ is an incorrect output class. This output class error does not change the number of copies to the original value. Neither does the number of copies or the class change from their current settings if the following command is issued:

```
print class $ copies 3
```

The output class value is again incorrect, so processing stops when this error is detected.

RECALL

```
►►──RECALL──┬──────────────────────┬────────────────────────────────────────────►◄
            ├─stored_statement_name─┤
            └─PREVIOUS──────────────┘
```

RECALL is an ISQL command that retrieves a stored SQL statement. The stored statement is inserted into the SQL command buffer and is shown in your display. You must enter the START command before the statement is started.

stored_statement_name
is the name of the stored SQL statement to be recalled.

PREVIOUS
specifies that the previous SQL statement (the one typed prior to the current SQL statement) is to be recalled. The current SQL statement is then saved with the name PREVIOUS.

Entering RECALL with no name specified, or pressing the appropriate PF key, displays the statement currently contained in the SQL command buffer. The RECALL command can be invoked by pressing PF5. You can assign the RECALL function to any PF key; the default is PF5 or PF17.

RENAME

```
►──RENAME──old_stored_statement_name──────────────────────────────────────►

►──new_stored_statement_name─────────────────────────────────────────────►◄
```

RENAME is an ISQL command that changes the name of a stored SQL statement.

old_stored_statement_name
: is the current name of the stored SQL statement. The name PREVIOUS should not be used.

new_stored_statement_name
: is the new name for the stored SQL statement. It can be up to 8 characters. Do not use the name PREVIOUS.

RIGHT

```
       ┌─1───────┐
►►─RIGHT─┼─────────┼──────────────────────────────────────────────►◄
       └─integer─┘
```

RIGHT is an ISQL display command that causes the display to start *integer* columns to the right, counting from the leftmost column of the display, provided no columns are being excluded. The number refers to the columns provided by the SELECT statement, not those currently displayed. You must use the order of the columns specified in the SELECT clause of the SELECT statement that provided the query result to determine the value for *integer*. If the value of *integer* represents a column being excluded, the display starts at the next displayable column to the right of the excluded column. If no more columns to the right can be displayed, a blank display results.

Specifying a value greater than the number of columns remaining to the right displays the last column or a blank display if the last column is excluded.

If no number is specified, the display moves one column to the right or to the next displayable column to the right if that column is excluded.

The RIGHT 1 command can be invoked by pressing PF11. Because you can reset your PF key values, you can assign the RIGHT 1 function to any PF key. The default function key is PF11 or PF23.

RUN

```
►►──RUN──routine_name──┬──────────────────────────┬──────────────────────────────►◄
                       │      ┌──────────┐        │
                       └─(──▼─parameter─┴───)─┘
```

RUN is an ISQL command that starts the processing of a routine.

You can run another user's routine if you have obtained the SELECT privilege by a GRANT statement on that user's ROUTINE table. Once you have the SELECT privilege on the user's ROUTINE table, you can run those routines by qualifying the routine name with the owner's authorization ID. For example, to run a routine called REPORTS that is owned by a user whose authorization ID is MIKE, you use:

```
run mike.reports
```

You do not have to identify the table name, ROUTINE, because the name of everyone's routine table is the same. Care must be taken in running another user's routine because any stored SQL statements or synonyms used in a routine are not recognized unless you have also defined them.

Parameters can also be passed to the routine by including them on the RUN command.

routine_name
is the name of the routine to be run.

parameter
is the parameter to be substituted for placeholders in the commands contained in the named routine. If more than one parameter is used, separate them by blanks.

Enclose a parameter in single quotation marks if it contains a blank. Any characters between the terminating single quotation mark and the next blank are ignored.

If there are more placeholders in the routine than there are parameters on the RUN command, the extra placeholders are replaced by null characters. Extra parameters on the RUN command are ignored.

Example: The following command executes your routine named SAMPLE and provides three parameters:

```
run sample (BLUE RED,GREEN 'WHITE BLACK')
```

The placeholders (indicated by &) in the routine are replaced by:

- BLUE to be substituted for all occurrences of &1
- RED,GREEN to be substituted for all occurrences of &2
- WHITE BLACK to be substituted for all occurrences of &3.

SAVE

```
►►──SAVE──────────────────────────────────────────────────────────────►◄
```

SAVE is an ISQL command that is only meaningful while using the INPUT command with AUTOCOMMIT on. If AUTOCOMMIT is on, it saves all changes made since the last SAVE command or since the start of the INPUT command if no SAVE command has been typed.

SAVE has no effect with AUTOCOMMIT off. With AUTOCOMMIT off, a COMMIT is necessary to save all input. This also commits all changes made during the current logical unit of work. The COMMIT must be typed after the INPUT command is ended.

SET

```
►►──SET──┤ Group_1 ├──┤ Group_2 ├──────────────────────────────────────────►◄
```

Group_1:

```
   ┌───────────────────────────────────────┐
   ▼          ┌─ON──┐                      │
├──┬─AUTOCommit─┤     ├──────────────────────┬─┴──────────────────────────────┤
   │          └─OFF─┘                      │
   │    ┌─UPper──┐                         │
   ├─CASE─┤        ├─────────────────────────┤
   │    └─STRing─┘                         │
   │          ┌─?───────────┐              │
   ├─CLAss(1)─┴─class(2)────┴──────────────┤
   │           ┌─hyphen────┐               │
   ├─CONTinue──┤           ├───────────────┤
   │           └─character─┘               │
   │           ┌─1──────────┐              │
   ├─COPies(1)─┤            ├──────────────┤
   │           └─integer(3)─┘              │
   │          ┌─0───────┐                  │
   ├─COSTest──┼─integer─┼──────────────────┤
   │          └─OFF─────┘                  │
   │          ┌─//./──────────────────┐    │
   └─DECimal──┤                       ├────┘
              └─/thousands/decimal/───┘
```

Notes:

1 The parameters CLAss, COPies, and PRINTRoute are not applicable to VM.

2 Output class wanted (letters from A to Z).

3 Number of copies to be printed.

4 A maximum of 20 characters can be used for null values.

Group_2:

```
   ┌──────────────────────────────────────────────────────┐
   ▼              ┌─RR─┐                                  │
├──┬─ISOLation────┼─CS─┼──────────────────────────────────┬┴─────────────────┤
   │              └─UR─┘                                  │
   ├─LANGuage──┬─language_name─┬──────────────────────────┤
   │           └─langid────────┘                          │
   │     ┌─?─────────┐                                    │
   ├─NULL┤           ├────────────────────────────────────┤
   │     └─string(4)─┘                                    │
   │                   ┌─132─────┐          ┌─66──────┐   │
   ├─PAGEsize──WIDth───┤         ├──LENgth──┤         ├───┤
   │                   └─integer─┘          └─integer─┘   │
   ├─PRINTRoute(1)──┬─DESTid──wkstat─┬────────────────────┤
   │                ├─TERMid──termid─┤                    │
   │                ├─TOUser──userid─┤                    │
   │                └─SYSTem─────────┘                    │
   │          ┌─CONTInue─┐                                │
   ├─RUNMode──┼─STOP─────┼────────────────────────────────┤
   │          └─CANCEL───┘                                │
   │              ┌─2───────┐                             │
   ├─SEParator──┬─┤         ├──BLANKs─┬───────────────────┤
   │            │ └─integer─┘         │                   │
   │            └─string──────────────┘                   │
   │          ┌─20──────┐                                 │
   └─VARChar──┤         ├─────────────────────────────────┘
              └─integer─┘
```

Notes:

1 The parameters CLAss, COPies, and PRINTRoute are not applicable to VM.

2 Output class wanted, (letters from A to Z).

3 Number of copies to be printed.

4 A maximum of 20 characters can be used for null values.

SET is an ISQL command that controls a number of functions.

When ISQL is started, the following operational characteristics are in force:

- AUTOCOMMIT is ON.
- CASE is UPPER.
- CLASS is a question mark (?). (DB2 Server for VSE only)
- CONTINUE is a hyphen (-).
- COPIES is 1. (DB2 Server for VSE only)
- COSTEST is zero.
- DECIMAL is //./.
- ISOLATION is repeatable read.
- LANGUAGE is the default language as defined in the SYSOPTIONS catalog.
- NULL is a question mark (?).
- PAGESIZE is LENGTH=66 WIDTH=132.
- PRINTROUTE is to the system printer. (DB2 Server for VSE only)
- RUNMODE is CONTINUE.
- SEPARATOR is two blanks.
- VARCHAR is 20.

The current setting of these functions can be listed with the LIST command. Some of these settings may already have been changed by a PROFILE routine, or you may want to change the settings for your session in your PROFILE routine, which is run automatically when you start ISQL.

Notes:

1. In the VM environment, the user must use the CP commands SPOOL and TAG to change the routing of the print output.
2. When you are using DRDA protocol in the VSE or VM environment, the isolation level is set to CS.

AUTOCommit
specifies if commands are to be committed automatically. Unless otherwise specified, commands are committed automatically.

ON
specifies that changes to tables resulting from an SQL statement are committed automatically when the statement is processed. This is the default.

There is an exception to the automatic committing of changes. If the SQL statement is INSERT, UPDATE, or DELETE, and it affects more than one data row, the changes are not immediately committed. Instead, the system issues a message that lets you either commit the work, or cancel or rollback the changes. If you type CANCEL or ROLLBACK, the changes are not committed; if you type any other statement, the changes are committed.

OFF
specifies that changes are not to be committed to a table until a COMMIT statement is typed.

.

CASE
specifies if characters enclosed in single quotation marks are to be converted to uppercase. Unless otherwise specified, all characters typed from the keyboard are converted to uppercase.

If you specify that the characters are to be converted, SQL uses the conversion information from the SYSCHARSETS system catalog to handle the conversion.

UPper
specifies that all characters typed from the keyboard or a routine are converted to uppercase characters. This is the default.

STRing
specifies that all characters enclosed in single quotation marks typed from the keyboard or a routine are not converted to uppercase characters.

CLAss (DB2 Server for VSE only)
specifies the printer output class.

class
is the output class desired (a letter from A to Z).

? specifies the default printer class of the system.

Unless otherwise specified, the default printer class for the system is used.

Note: Class cannot be specified for a CICS/VSE terminal printer.

CONTInue
specifies the continuation character.

character
is the continuation character to use for continuation of input lines.

Choose a character that is not normally the last character of a statement. Also, do not use a single or double quotation mark, a semicolon, or a blank. Unless otherwise set, the continuation character is the hyphen (-).

COPies (DB2 Server for VSE only)
specifies the number of copies for printed reports.

integer
is the number of copies that are to be printed when subsequent PRINT commands are issued.

Unless otherwise specified, the number of copies for printed reports is one. A maximum of 99 copies can be specified.

Note: The number of copies cannot be specified for a CICS/VSE terminal printer.

COSTest
specifies when the ISQL *Query Cost Estimate* (QCE) is displayed. Unless OFF is specified, the QCE message is always displayed.

OFF
specifies that the ISQL QCE message should not be displayed.

integer
specifies that the ISQL QCE message should be displayed. Replace *integer* with any number from 0 to 9999. The QCE is displayed if it is greater than *integer*.

While you cannot specify any number greater than 9999, the QCE message displays ‘>9999’ to indicate the cost is greater than 9999.

For additional information about the ISQL Query Cost Estimate, refer to the *DB2 Server for VSE Database Administration* and *DB2 Server for VM Database Administration* manuals.

DECimal
specifies the type of punctuation to use when displaying a decimal column.

/thousands/
is the thousands separator character. Valid characters are a period, a comma, and a blank.

/decimal/
is the decimal separator character. Valid characters are a period and a comma.

The slash distinguishes the thousands separator character from the decimal separator character.

Unless otherwise specified, no thousands separator is used, and the decimal separator is a period.

Valid combinations of t and d are:

Thousands Separator	Decimal Separator	Example
(nothing)	.	1234.56
,	.	1,234.56
.	,	1.234,56
(a blank)	,	1 234,56

For example, assume the value *123456* is contained in a decimal column that was created with two decimal places. Then, specifying:

```
set decimal /,/./
```

provides the following punctuation when this field is displayed:

```
1,234.56
```

The DECIMAL keyword does not change how input is specified, only how output is displayed. For example, to reference the above number (1,234.56) in a column called SALES in a WHERE clause, you use:

```
where SALES=1234.56
```

.

ISOLation
specifies the isolation level placed on data when you read information from the application server. You specify the type of lock that the system places on the data. This is specifying the *isolation level*. The isolation level specified sets the degree of independence one terminal user has from another terminal user.

For guidelines on using this setting, see “Specifying the Isolation Level” on page 194.

RR
requests the isolation level repeatable read. This setting holds locks on the data you are using until a COMMIT or ROLLBACK is performed. These locks isolate the data from other users. No one can modify any rows you have read until your work has been committed or rolled back. Use this

setting when it is important to keep data completely isolated. This is the default.

Note: For DB2 Server for VSE & VM, if you are using the DRDA protocol, the default isolation level is set to CS. If you specify any other setting, it will be ignored.

CS

requests the isolation level cursor stability. Isolation level cursor stability has meaning only for data in public dbspaces with row and page level locking. The system locks individual rows or pages depending on the lock specified in ACQUIRE DBSPACE, ALTER DBSPACE, and LOCK statements. Use the CS setting to free the data you are reading as soon as possible.

UR

requests the isolation level uncommitted read. Isolation level uncommitted read has meaning only for data in public dbspaces with row and page level locking. This setting applies only to read-only operations (SELECTs). For other operations (UPDATE, DELETE, and INSERT), the rules of CS apply. This setting reduces lock contention on data being read; however, data integrity may be compromised because read-only access to uncommitted data is allowed. Use the UR setting only when it is not necessary that the data you are reading be committed.

The isolation level specification affects the UPDATE, DELETE, INSERT with subselect form, and SELECT statements. The correct value to specify depends on what activity you are performing. Locking problems can be reduced if the isolation level can be set to cursor stability in your system.

Note: Read and update access to the catalog is performed with a repeatable read setting, regardless of how you set the isolation level. This access is activated by dynamic preprocessing of SQL statements and by SQL data definition statements such as CREATE, ACQUIRE, and GRANT. Your selects (and updates and deletes, if you are a DBA) against the catalog are performed according to the isolation level you set.

SET ISOLATION is acceptable only when the target application server is a local application server. Otherwise the isolation level of CS is assumed and the SET ISOLATION command will have no effect.

LANGuage

specifies the language in which online HELP and error messages are displayed. Operator messages are displayed in the national language of the application server.

language_name

is the language being specified; for example, French. The description of the language can be either the IBM-supplied description or the description chosen by your site. For example, your site may prefer to use *Francais* instead of *French*. *Language-name* can be up to 40 characters long.

langid

is a 5-character language identifier that can be specified instead of the language name. The language identifiers are:

AMENG Mixed Case American English

UCENG	Uppercase American English
FRANC	French
GER	German
KANJI	Japanese
KOR	Korean
HANZI	Simplified Chinese

If the language you specify is not supported, the current language remains unchanged.

NULL
specifies the characters to be displayed in null fields. Unless otherwise specified, a question mark (?) is used.

string
specifies the characters (up to a maximum of 20) to use for null fields. Enclose the string in single quotation marks if it contains a blank.

For example, the following command:

```
set null empty
```

causes the word EMPTY to be displayed in all null fields.

PAGEsize
specifies the page size for printed output.

WIDth
specifies the width of the paper being used.

integer
is the number of characters that can fit on a line of the output paper. Unless otherwise specified, the page width is 132. You can specify values from 19 to 204.

Note: If you are sending your output to the terminal printer, you should set the page width to the printer width-1 to avoid spacing problems. For example, if the terminal printer width is 132, then set the width to 131 with the command SET PAGE WID 131.

LENgth
specifies the length of the page being used.

integer
is the number of printed lines that can fit on the output paper. Unless otherwise specified, the page length is 66. You can specify values from 9 to 32767. The maximum number of lines that can actually be printed on a page is 8 less than the length. Eight lines are reserved for top and bottom titles and margins.

PRINTRoute (DB2 Server for VSE only)
specifies where print output is to be sent.

DESTid
specifies that printed output is to be directed to the designated POWER remote workstation.

wkstat
is the ID of the desired remote workstation. *wkstat* can be any number from 0 to 250. When *wkstat* is 0, the printed output is to be directed to the system printer.

TERMid
specifies that printed output is to be directed to the designated CICS/VSE terminal.

termid
is the terminal identifier of the desired CICS/VSE terminal. The *termid* must be from one to four alphanumeric characters.

TOUser
spools the print output the same way it will spool the print output when the user enters `PRINT TOUser` *`userid`*

userid
is the VSE POWER user identifier of the user to whom the output is being spooled. An identifier cannot be longer than eight alphanumeric characters. If the *userid* is any number from 1 to 250, ISQL will spool the print output to the POWER remote workstation whome ID is the number specified.

SYStem
specifies that printed output is to be directed to the system printer.

RUNMode
specifies whether processing should continue when an error is detected in a routine. Unless otherwise specified, processing continues.

CONTInue
specifies that processing is to continue to the next command even if errors are detected in the routine. This is the default.

STOP
specifies that processing is to stop if an error is detected in the routine. No rollback is performed, and processing is terminated.

CANCEL
specifies that an internal CANCEL is to be issued if an error is detected. A rollback is issued internally.

SEParator
specifies the separation between columns. Unless otherwise specified, two blanks are used.

integer **BLANKs**
specifies the number of spaces to be displayed between columns. Replace *integer* with the number of blanks desired. The maximum number of blanks that can be specified is 254.

string
specifies the characters to be displayed between columns. Enclose the string in single quotation marks if it contains a blank. For example, if you want to draw a vertical line between columns, you type:

```
set separator ' | '
```

which places a blank, a vertical bar, and a blank between all columns. The string can be up to 254 characters long.

VARChar
specifies the display width of variable length columns.

integer
is the length desired. The maximum width that can be specified is 254. Unless otherwise set, the VARCHAR length is 20. Since only the first 20 characters of a variable length column are displayed, this command or a FORMAT VARCHAR is necessary to view those columns that are wider than 20 characters.

The SET command can be used with more than one keyword option, allowing you to set several operational characteristics with a single SET command. These characteristics are effective for the duration of your terminal session. In the following example, a single SET command sets operational characteristics for AUTOCOMMIT, NULL, and SEPARATOR:

```
set autocommit on null 'no data' separator 2 blanks
```

SHOW

```
>>──SHOW──┬─ACTIVE──────────────────────────────────────────────────┬──────────><
          ├─ADDRESS──module-name────────────────────────────────────┤
          ├─BUFFERS─────────────────────────────────────────────────┤
          │          ┌─ALL──────────────────────┐                    │
          ├─CONNECT──┼──────────────────────────┼────────────────────┤
          │          ├─┬────────┬──userid1──────┤                    │
          │          │ └─USERID─┘               │                    │
          │          ├─AGENT──agent-no──────────┤                    │
          │          ├─LUWID──luwid1────────────┤                    │
          │          ├─ACTIVE───────────────────┤                    │
          │          ├─INACTIVE─────────────────┤                    │
          │          └─WAITING─(1)──────────────┘                    │
          ├─DBCONFIG────────────────────────────────────────────────┤
          ├─DBEXTENT────────────────────────────────────────────────┤
          ├─DBSPACE──dbspace-number─────────────────────────────────┤
          ├─INVALID─────────────────────────────────────────────────┤
          ├─LOG─────────────────────────────────────────────────────┤
          │          ┌─ALL─────┐                                     │
          ├─LOGHIST──┼─────────┼──┬─────────┬───────────────────────┤
          │          └─integer─┘  └─SERVICE─┘                        │
          │       ┌─MATRIX──────────────────────────────────────┐    │
          ├─LOCK──┼─────────────────────────────────────────────┼────┤
          │       ├─ACTIVE──────────────────────────────────────┤    │
          │       ├─DBSPACE──┬─ALL────────────┬─────────────────┤    │
          │       │          └─dbspace_number─┘                 │    │
          │       ├─GRAPH──┬─┬────────┬──authorization_name─┬───┤    │
          │       │        │ └─USERID─┘                     │   │    │
          │       │        └─AGENT──agent_number────────────┘   │    │
          │       │       ┌─ALL──────────────────────────┐      │    │
          │       ├─USER──┼──────────────────────────────┼──────┤    │
          │       │       ├─┬────────┬──authorization_name─┤    │    │
          │       │       │ └─USERID─┘                   │      │    │
          │       │       └─AGENT──agent_number──────────┘      │    │
          │       │           ┌─ALL──────────────────────────┐  │    │
          │       └─WANTLOCK──┼──────────────────────────────┼──┘    │
          │                   ├─┬────────┬──authorization_name─┤     │
          │                   │ └─USERID─┘                   │       │
          │                   └─AGENT──agent_number──────────┘       │
          │       ┌─ALL──────────┐                                   │
          ├─POOL──┼──────────────┼───────────────────────────────────┤
          │       ├─SUMMARY──────┤                                   │
          │       ├─DELETED──────┤                                   │
          │       └─pool_number──┘                                   │
          ├─STORAGE─────────────────────────────────────────────────┤
          ├─SYSTEM──────────────────────────────────────────────────┤
          └─USERS─(1)───────────────────────────────────────────────┘
```

Note:
[1] The parameters WAITING and USERS are not applicable to DB2 Server for VSE

SHOW is an operator command used primarily to monitor system activity. It is not allowed during an LUW, and you must end an LUW that is in progress before issuing SHOW.

| The SHOW command can only be used when the target application server is a
| local application server. It cannot be used when the application server is a remote
| application server.

See the *DB2 Server for VSE & VM Operation* manual for a description of the keywords used on this command and for a description of the information displayed.

This command results in one or more displays.

DB2 Server for VSE

Instructions for proceeding to the next display of data or ending the display are provided in the status area.

DB2 Server for VM

Instructions for proceeding to the next display of data or ending the display are provided in the output area.

When using the SHOW DBSPACE command, replace *dbspace-number* with the number of the particular dbspace for which you want information.

The SHOW command is not available on non-DB2 Server for VSE & VM application servers or if you are using the DRDA protocol.

Note: If the national language of the application server differs from the national language that the user has set for the ISQL session, the messages generated by this operator command are issued in the national language of the application server.

START

```
>>──START──┬──────────────────────┬──┬───────────────────────┬──────────────────────────><
           └─stored_statement_name─┘  │     ┌──────────┐      │
                                      └─(──▼─parameter─┴──)─┘
```

START is an ISQL command that causes a stored SQL statement or the current SQL statement to be processed.

The START command can be invoked by pressing PF2. The START command can be invoked by pressing PF2 (or PF14). Because you can reset your PF keys, you can assign the START function to any PF key.

stored_statement_name
: is the name of the stored SQL statement to be started. The name PREVIOUS is not allowed. To start the previous SQL statement, you must first recall it to the command buffer and enter START without supplying a name. If *stored_statement_name* is not specified, or START PF is pressed, the current SQL statement is started.

parameter
: is the parameter to be substituted for the placeholders in the SQL statement. If you use more than one parameter, separate them with blanks. Enclose a parameter in single quotation marks if it contains a blank. Any characters between the terminating single quotation mark and the next blank are ignored.

  All placeholders in the SQL statement that do not have a corresponding parameter on the START command are replaced by null characters or are erased. Extra parameters on the START command are ignored.

  The parameter can contain DBCS characters.

Example: Assume you have the following SQL statement stored as *SELPD*:

```
select &1,&2,&3 -
from emp_act -
where actno = &4 -
  or emstdate = '&5'
```

You could type the START command as:

```
start selpd (empno actno emptime 90 '1982-06-01')
```

The resulting statement started is:

```
select empno,actno,emptime -
from emp_act -
where actno = 90 -
  or emstdate = '1982-06-01'
```

Notice that single quotation marks are needed around the &5 placeholder in the stored SQL statement because that placeholder stands for a character data item. Single quotation marks are not needed around &4 because &4 stands for a numeric data item. Single quotation marks are not needed for &1, &2, and &3 because they do not contain any blanks.

In a stored SQL statement, you can only use the ampersand (&) to create placeholders.

In the above example, the formatting information is not saved because placeholders are used in the SELECT clause.

STORE

```
►►──STORE──stored_statement_name──┬─────────┬──────────────────────────────►◄
                                  └─REPLACE─┘
```

STORE is an ISQL command that causes the current SQL (only) statement to be saved for later use. SQL statements remain stored until erased. When storing an SQL statement, you must associate a name with it. Use this name to recall the SQL statement when you want to execute it. The SQL statement, when stored, also remains as the current SQL statement.

Stored statements can be started, listed, renamed, erased, or recalled. When a stored statement is started, renamed, or recalled, it becomes the current SQL statement.

stored_statement_name
: is the name you want to use to refer to the stored statement. Names can be up to 8 characters long. The name PREVIOUS is not allowed because ISQL always stores the current statement under that name when a new SQL statement is typed.

REPLACE
: specifies that the current statement is to replace any existing stored SQL statement with the same name. If REPLACE is not specified and there is an existing stored SQL statement with the same name, a warning message is issued. This message gives you three options to select from:

- REPLACE, to replace the existing stored SQL statement with the current SQL statement
- END, to end the processing of the STORE command.
- Type a different name to be used as the name for the current SQL statement being stored.

When you store a SELECT statement, related display formatting information is also stored. Formatting is defined by FORMAT commands or current DB2 Server for VSE & VM formatting defaults. This information, which cannot be “seen”, remains stored with the SELECT statement and formats the display when the statement is recalled and executed. However, the following exceptions exist:

- Formatting information is not saved when SELECT statements are stored that contain placeholders in the SELECT or FROM clauses. Formatting information *is saved* when the placeholders occur in clauses other than the SELECT and FROM clauses.
- If a stored SELECT command is changed by a CHANGE command, formatting information is saved only when the change occurs in the WHERE, GROUP BY, ORDER BY, or HAVING clauses (that is, when the change does not occur in the SELECT or FROM clauses).
- If a table referenced by a stored query is dropped and later created with different column characteristics (such as DECIMAL instead of INTEGER), existing formatting information is not used.

The notation *SELECT * FROM table_name* is not recommended for use in stored SELECT statements. It is possible that someone can add a new column to the table, which causes any stored format information for the SELECT statement to be erased. By explicitly naming the columns required, or by referring to a view instead of a table (the view can have SELECT *), you can avoid this problem.

The steps necessary to store FORMAT command information along with an SQL statement are:

1. Execute a SELECT statement to retrieve data.
2. Type one or more FORMAT commands to format the display.
3. Type an END command to end the display.
4. Type a STORE command to store the statement (which is still the current SQL statement) and any formatting information.

TAB

```
         ┌─1───────┐
►►──TAB──┼─────────┼──────────────────────────────────────────────────►◄
         └─integer─┘
```

TAB is an ISQL display command that enables you to view all the characters of a column that is too wide to fit on the display. It lets you display any adjacent characters of the column; the number of characters displayed is equal to the width of your display. Before issuing TAB, place the desired column at the left edge of the display.

integer
is the number that represents the character's position where the display is to start. If no number is specified, the display starts at the first character position of the column.

The TAB command is valid only for CHAR or VARCHAR columns.

Example

Suppose you are viewing a column whose length attribute is 100 characters. If you are using a 24 x 80 display and you want to view the characters beyond the 80th, you type:

```
tab 81
```

This displays the column, starting with the 81st character at the left edge of the display.

Appendix A. Answers to the Exercises

The following answers are shown with each clause on a separate line so that you can easily check your commands. You can, of course, put the entire command on one line if it fits. Or, you can use several lines and break each line at a different place than shown here. It is your choice. Just follow the rules for using the continuation character.

Exercise 1:

(page 24)

1.
```
select actno -
from activity
```

(Remember to type an END command to end the query.)

2.
```
select deptno,respemp,prstaff -
from project
```

3.
```
select * -
from proj_act
```

Exercise 2:

(page 30)

1.
```
select lastname,workdept,phoneno -
from employee -
where salary>50000.00
```

2.
```
select * -
from department -
where admrdept='A00'
    OR
select deptno,deptname,mgrno,admrdept -
from department -
where admrdept='A00'
```

3.
```
select actno -
from proj_act -
where acstaff>=0.75
```

4.
```
select * -
from proj_act -
where projno='ad3100' or projno='if1000' or projno='ma2111'
    OR
select projno,actno,acstaff,acstdate,acendate -
from proj_act -
where projno='ad3100' or projno='if1000' or projno='ma2111'
```

Exercise 3:

(page 37)

1.
```
select actno -
from proj_act -
where acstaff between 0.75 and 1.25
```

2.
```
select projno,acstdate -
from proj_act -
where projno in ('ad3100','if1000','ma2111')
```

3.
```
select actno -
from activity -
where actdesc like '%data%'
```

Exercise 4:

(page 43)

1.
```
select empno,lastname,salary*1.10 -
from employee -
where bonus>=1000
```

2.
```
select projno,actno -
from proj_act -
where acstaff+0.5<1.5
```

Exercise 5:

(page 56)

1.
```
select sum(acstaff*0.75) -
from proj_act -
where actno=20
```

2.
```
select count(distinct deptno) -
from project
```

3.
```
select min(salary) -
from employee -
where workdept='D11'
```

Exercise 6:

(page 59)

1.
```
select * -
from proj_act -
where projno='op2010' -
order by acstaff,actno
    OR
select projno,actno,acstaff,acstdate,acendate -
from proj_act -
where projno='op2010' -
order by 3,2
```

2.
```
select projno,projname,'ends Feb 1, 1983' -
from project -
where prendate='1983-02-01' -
order by projno desc
```

Exercise 7:

(page 69)

```
1.   forward 39
2.   forward max
3.   right 1     or    column 2
4.   left 1      or    column 1
5.   backward max
6.   print
7.   end
```

Exercise 8:

(page 85)

```
1.   set autocommit off

2.   insert into activity (actno,actkwd,actdesc) -
     values (190, -
             'market', -
             'marketing')

3.   rollback

4.   select * -
     from project -
     where majproj is null

5.   update project -
     set prstaff=prstaff+0.5,prendate=prendate-7 days
         OR
     update project -
     set prstaff=prstaff+0.5
         AND
     update project -
     set prendate=prendate-7 days

6.   update project -
     set prstdate='1982-03-01' -
     where deptno='e21' and majproj='op2010'

7.   input department
     'f01','personnel','000110','a00'
     'g01','marketing and sales','000120','a00'
     end

8.   delete from department -
     where deptno='f01' or deptno='f11' or deptno='f21'
         OR
     delete from department -
     where deptno in ('f01','f11','f21')

9.   a.  select * -
         from department -
         order by deptno

     b.  select * -
         from project -
         order by projno
```

```
10.     insert into employee -
        values ('000060','IRVING','F','STERN','D11', -
        '6423','1973-09-14','manager',20,'m', -
        '1953-01-01',45000.00,2000.00,800.00)

11.     delete from project -
        where projno='MA2114'
         OR
        delete from project -
        where projname='Expert Systems'

12.     a.   select * from employee
        b.   select * from project

13.     a.   rollback
        b.   set autocommit on
```

Exercise 9:

(page 94)

```
1.   hold select * from department
2.   change /*/&1,&2/;
3.   a.   start (deptno deptname)
     b.   end
4.   change /nt/nt where &3/;
5.   start (deptname mgrno admrdept='e01')
```

Exercise 10:

(page 102)

```
1.   select empno,projno,emptime -
     from emp_act -
     where projno='if1000' or projno='if2000' -
     order by projno,empno

2.   format separator '  *  '

3.   format exclude empno
         OR
     format exclude 1
         OR
     format include only (projno emptime)
         OR
     format include only (2 3)

4.  a.  format column emptime name proptn
            OR
        format column 3 name proptn

    b.  format column proptn width 8
            OR
        format column 3 width 8
```

Note: You can also do items 2 through 4 using a single FORMAT command:

```
format separator ' * ' exclude 1 -
column emptime name proptn width 8
```

Exercise 11:

(page 117)

DB2 Server for VSE

1. `set copies 2`
2. `list set *`
3. ```
   select * -
   from proj_act -
   where projno='ad3112' -
   order by acendate
   ```
4. ```
   format group acendate
       AND
   format exclude (acstdate)
       OR
   format group acendate exclude (4)
   ```
5. `format ttitle 'personnel programming deadlines'`
6. `print`

DB2 Server for VM

1. `list set *`
2. ```
   select * -
   from proj_act -
   where projno='ad3112' -
   order by acendate
   ```
3. ```
   format group acendate
       AND
   format exclude (acstdate)
       OR
   format group acendate exclude (4)
   ```
4. `format ttitle 'personnel programming deadlines'`
5. `print copies 2`

Exercise 12:

(page 124)

```
1.  recall myquery
2.  change /&1/salary between 25000 and 30000 order by 2/
3.  start
4.  format exclude (3)
        OR
    format exclude (midinit)
5.  format separator ' |*| '
6.  format column edlevel name 'school years'
7.  a.  end
    b.  store EXER11
8.  list sql *
9.  help store
```

Exercise 13:

(page 133)

This answer assumes you have a REMARKS column in your ROUTINE table.

```
1.   input routine
     a.  'exer13',10,'select actno,actdesc -',null
         'exer13',20,'from activity',null
     b.  'exer13',30,'format separator 3 blanks',null
     c.  'exer13',40,'display',null
     d.  'exer13',50,'print copies 3',null
     e.  'exer13',60,'end',null
     end
```

Note: Any sequence numbers are valid as long as they are in ascending order.

Exercise 14:

(page 150)

```
1.   select ea.actno,empno,projno,emptime,actdesc -
     from emp_act ea,activity a -
     where ea.actno=a.actno -
     order by actno

2.   select actno,sum(emptime) -
     from emp_act -
     group by actno -
     order by actno

3.   create view empls (activity,fraction_time,surname) -
     as select actno,emptime,lastname -
        from emp_act emac,employee em -
        where emac.empno=em.empno

4.   select * -
     from empls -
     where activity=100

5.   select * -
     from empls -
     order by activity,fraction_time

6.   format group activity ttitle 'who is doing what'
         OR
     format group activity
     format ttitle 'who is doing what'

7.   select * -
     from proj_act -
     where projno = 'AD3113' -
     order by actno

8.   select actno,acstaff -
     from proj_act -
     where acendate='1983-01-01' -
     and actno > (select max(actno) -
                  from proj_act -
                  where projno='if2000')
```

9.
```
select lastname,salary -
from employee -
where job='designer' -
and salary > (select avg(salary) -
             from employee)
```

Exercise 15:

(page 173)

1.
```
select pa.projno,projname,prstaff,actno,acstaff -
from proj_act pa,project p -
where pa.projno=p.projno -
      and 0.5*prstaff < (select sum(acstaff) -
                         from proj_act -
                         where pa.actno=actno) -
order by pa.projno
```

2.
```
select p.projno,p.actno,acstaff,emptime,empno -
from proj_act p,emp_act e -
where 2*emptime < (select avg(acstaff) -
                     from proj_act -
                     where projno=p.projno) -
           and 3 <= (select count(*) -
                     from proj_act -
                     where projno=p.projno)
```

3.
```
select sum(comm),'manufacturing systems' -
from employee -
where workdept='d11' -
union -
select sum(comm),'administration systems' -
from employee -
where workdept='d21'
```

4.
```
select empno,actno,emptime -
from emp_act -
where projno = (select projno -
                from project -
                where projname='w l programming') -
union -
select empno,actno,emptime -
from emp_act -
where projno='ma2112'
```

5.
```
create view managers -
as select * -
from employee -
where job='manager'
```

6.
```
select lastname,projno,actno -
from managers m,emp_act e -
where m.empno=e.empno
```

7.
```
select avg(salary),max(salary), -
min(salary),'managers' -
from managers -
union -
select avg(salary),max(salary), -
min(salary),'employees' -
from employee
```

8. drop view managers

9. select viewname -
   from system.sysviews -
   where vcreator=user
       OR
   select tname -
   from system.syscatalog -
   where creator=user and tabletype='v'

10. drop view act10
    drop view act60
    drop view proj1
    drop view proac
    drop view proj3

11. delete from proj_act -
    where (actno=50 -
    and projno in ('ma2110','ad3112','if1000'))

Exercise 16:

(page 196)

```
1.  create table old_activity -
    (anumber smallint not null, -
     akeyword varchar(10) not null, -
     adescr varchar(30) not null)

2.  insert into old_activity (anumber,akeyword,adescr) -
       select * from sqldba.activity

3.  alter table old_activity -
    add status varchar(8)

4.  grant update(status) -
    on old_activity -
    to mona,lisa

5.  a.  create view farleyview -
        as select * -
        from old_activity -
        where status='inactive'

    b.  grant select -
        on farleyview -
        to farley
```

Appendix B. Sample Tables

The sample tables illustrated in this appendix are used in examples throughout the library. These tables simulate a database created for use in organization or project management applications. As a group, the tables include information that describes employees, departments, projects, and activities. Figure 119 shows the relationships among the tables. These relationships are established by referential constraints, where a foreign key in the dependent table references a primary key in the parent table. In the figure, the referential constraint is symbolized by lines joining the keys; the arrowheads point from the primary key to the foreign key. Only those columns named as foreign or primary keys are listed in the figure. All tables have additional columns. You can easily review the contents of any table by executing an SQL statement, such as `SELECT * FROM SQLDBA.DEPARTMENT`.

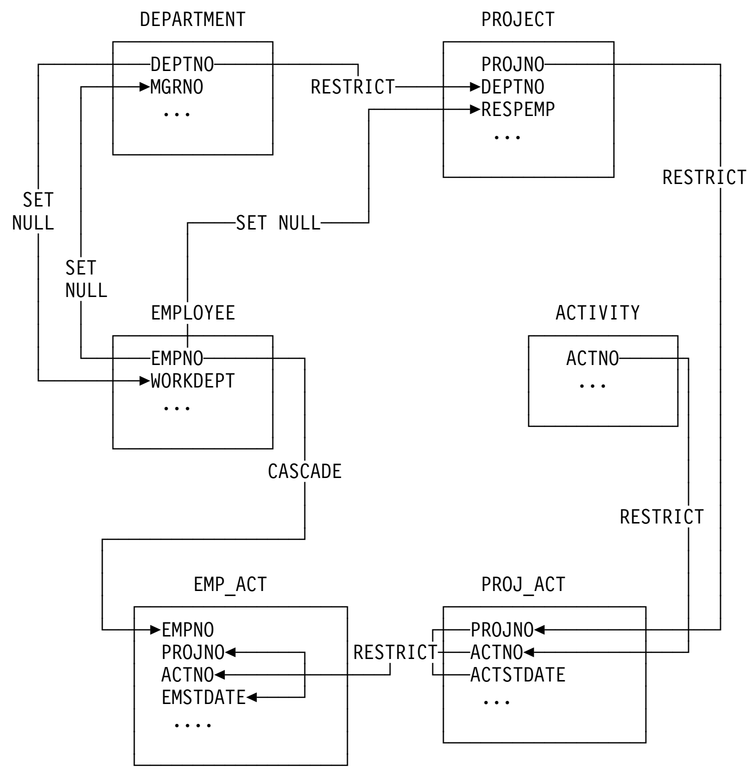


Figure 119. Relationships among Tables in the Sample Application

DEPARTMENT Table

The DEPARTMENT table describes each department in the business and identifies its manager and the department to which it reports. The table contents are shown in Figure 120 on page 270; a description of the columns is shown in Figure 121.

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	?	A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

Figure 120. DEPARTMENT Table Contents

Column Name	Description
DEPTNO	Department number, the primary key
DEPTNAME	A name describing the general activities of the department
MGRNO	Employee number (EMPNO) of the department manager
ADMRDEPT	Number of the department to which this department reports; the department at the highest level reports to itself

Figure 121. Columns of the DEPARTMENT Table

The DEPARTMENT table is created with:

```
CREATE TABLE DEPARTMENT
    (DEPTNO    CHAR(3)          NOT NULL,
     DEPTNAME  VARCHAR(36)      NOT NULL,
     MGRNO     CHAR(6)                  ,
     ADMRDEPT  CHAR(3)          NOT NULL,
     PRIMARY KEY (DEPTNO)               )
```

After the EMPLOYEE table has been created, a foreign key is added to the DEPARTMENT table with this statement:

```
ALTER TABLE DEPARTMENT ADD
     FOREIGN KEY R_EMPLY1 (MGRNO) REFERENCES EMPLOYEE
          ON DELETE SET NULL
```

Relationship to Other Tables

DEPARTMENT is a parent of the EMPLOYEE and PROJECT tables.

The DEPARTMENT table is a dependent of the EMPLOYEE table; the MGRNO column is the foreign key in the DEPARTMENT table and references EMPNO, the primary key in the EMPLOYEE table.

EMPLOYEE Table

The EMPLOYEE table identifies all employees by an employee number and lists basic personnel information. The table in Figure 122 shows the contents of the EMPLOYEE table; Figure 123 on page 274 shows a description of the columns.

Figure 122 (Page 1 of 2). EMPLOYEE Table Contents

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(6) not null	varchar(12) not null	char(1) not null	varchar(15) not null	char(3)	char(4)	date	char(8)	smallint not null	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707

Figure 122 (Page 2 of 2). EMPLOYEE Table Contents

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

Figure 123. Columns of the EMPLOYEE Table	
Column Name	**Description**
EMPNO	Employee number (the primary key)
FIRSTNME	First name of the employee
MIDINIT	Middle initial of the employee
LASTNAME	Last name of the employee
WORKDEPT	Number of department in which the employee works
PHONENO	Employee telephone number
HIREDATE	Date of hire
JOB	Job held by the employee
EDLEVEL	Number of years of formal education
SEX	Sex of the employee (M or F)
BIRTHDATE	Date of birth
SALARY	Yearly salary
BONUS	Yearly bonus
COMM	Yearly commission

The EMPLOYEE table has a foreign key referencing the primary key in the DEPARTMENT table. The DEPARTMENT table must, therefore, be created first. The EMPLOYEE table is then created with:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)         NOT NULL,
    FIRSTNME  VARCHAR(12)     NOT NULL,
    MIDINIT   CHAR(1)         NOT NULL,
    LASTNAME  VARCHAR(15)     NOT NULL,
    WORKDEPT  CHAR(3)                 ,
    PHONENO   CHAR(4)                 ,
    HIREDATE  DATE                    ,
    JOB       CHAR(8)                 ,
    EDLEVEL   SMALLINT        NOT NULL,
    SEX       CHAR(1)                 ,
    BIRTHDATE DATE                    ,
    SALARY    DECIMAL(9,2)            ,
    BONUS     DECIMAL(9,2)            ,
    COMM      DECIMAL(9,2)            ,
    PRIMARY KEY (EMPNO)               ,
    FOREIGN KEY R_DEPT1 (WORKDEPT) REFERENCES DEPARTMENT
          ON DELETE SET NULL       )
```

Relationship to Other Tables

The EMPLOYEE table is a parent of the DEPARTMENT table, the PROJECT table, and the EMP_ACT table.

The EMPLOYEE table is a dependent of the DEPARTMENT table; the foreign key on the WORKDEPT column in the EMPLOYEE table references the primary key on the DEPTNO column in the DEPARTMENT table.

PROJECT Table

The PROJECT table describes each project that the business is currently undertaking. Data contained in each row includes the project number, name, person responsible, and schedule dates as shown in Figure 124; Figure 125 on page 276 describes the columns.

Figure 124. PROJECT Table Contents

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	?
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	?
IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	?
MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	?
MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	?
OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	?
OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

Figure 125. Columns of the PROJECT Table	
Column Name	**Description**
PROJNO	Project number (the primary key)
PROJNAME	Project name
DEPTNO	Number of department responsible for the project
RESPEMP	Number of employee responsible for the project
PRSTAFF	Estimated mean project staffing (mean number of persons) needed between PRSTDATE and PRENDATE to achieve the whole project, including any subprojects
PRSTDATE	Estimated project start date
PRENDATE	Estimated project end date
MAJPROJ	Number of any major project of which the subject project may be a part

The PROJECT table has foreign keys referencing DEPARTMENT and EMPLOYEE. The EMPLOYEE and DEPARTMENT tables must be created before the PROJECT table. Once EMPLOYEE and DEPARTMENT are created, the following statement creates the PROJECT table:

```
CREATE TABLE PROJECT
    (PROJNO    CHAR(6)        NOT NULL,
     PROJNAME  VARCHAR(24)    NOT NULL,
     DEPTNO    CHAR(3)        NOT NULL,
     RESPEMP   CHAR(6)                ,
     PRSTAFF   DECIMAL(5,2)           ,
     PRSTDATE  DATE                   ,
     PRENDATE  DATE                   ,
     MAJPROJ   CHAR(6)                ,
     PRIMARY KEY (PROJNO)             ,
     FOREIGN KEY R_DEPT2 (DEPTNO) REFERENCES DEPARTMENT
          ON DELETE RESTRICT          ,
     FOREIGN KEY R_EMPLY2 (RESPEMP) REFERENCES EMPLOYEE
          ON DELETE SET NULL          )
```

Relationship to Other Tables

PROJECT is a parent of the PROJ_ACT table.

PROJECT is a dependent of:

- The DEPARTMENT table; the foreign key on the DEPTNO column in PROJECT references the primary key in the DEPARTMENT table.
- The EMPLOYEE table; the foreign key on the RESPEMP column in PROJECT references the primary key in the EMPLOYEE table.

ACTIVITY Table

The ACTIVITY tables describes the activities that can be performed during a project. The table acts as a master list of possible activities, identifying the activity number, and providing a description of the activity. Figure 126 shows table contents; Figure 127 shows a description of the columns.

ACTNO	ACTKWD	ACTDESC
160	ADMDB	Adm databases
170	ADMDC	Adm data comm
90	ADMQS	Adm query system
150	ADMSYS	Adm operating sys
70	CODE	Code programs
110	COURSE	Develop courses
30	DEFINE	Define specs
180	DOC	Document
20	ECOST	Estimate cost
40	LEADPR	Lead program/design
60	LOGIC	Describe logic
140	MAINT	Maint software sys
10	MANAGE	Manage/advise
130	OPERAT	Oper computer sys
50	SPECS	Write specs
120	STAFF	Pers and staffing
100	TEACH	Teach classes
80	TEST	Test programs

Figure 126. ACTIVITY Table Contents

Column Name	Description
ACTNO	Activity number (the primary key)
ACTKWD	Activity keyword (up to six characters)
ACTDESC	Activity description

Figure 127. Columns of the ACTIVITY Table

The ACTIVITY table is created with:

```
CREATE TABLE ACTIVITY
     (ACTNO    SMALLINT       NOT NULL,
      ACTKWD   CHAR(6)        NOT NULL,
      ACTDESC  VARCHAR(20)    NOT NULL,
      PRIMARY KEY (ACTNO)             )
```

Relationship to Other Tables

ACTIVITY is a parent of the PROJ_ACT table.

PROJ_ACT Table

The PROJ_ACT table lists the activities performed for each project. The table contains information on the start and completion dates of the project activity as well as staffing requirements as shown in Figure 128. Figure 129 on page 279 shows a description of the columns.

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	0.50	1982-01-01	1982-07-01
AD3110	10	1.00	1982-01-01	1983-01-01
AD3111	60	0.80	1982-01-01	1982-04-15
AD3111	70	1.50	1982-02-15	1982-10-15
AD3111	80	1.25	1982-04-15	1983-01-15
AD3111	180	1.00	1982-10-15	1983-01-15
AD3112	60	0.75	1982-01-01	1982-05-15
AD3112	60	0.75	1982-12-01	1983-01-01
AD3112	70	0.75	1982-01-01	1982-10-15
AD3112	80	0.35	1982-08-15	1982-12-01
AD3112	180	0.50	1982-08-15	1983-01-01
AD3113	60	0.75	1982-03-01	1982-10-15
AD3113	70	1.25	1982-06-01	1982-12-15
AD3113	80	1.75	1982-01-01	1982-04-15
AD3113	180	0.75	1982-03-01	1982-07-01
IF1000	10	0.50	1982-01-01	1983-01-01
IF1000	90	1.00	1982-01-01	1983-01-01
IF1000	100	0.50	1982-01-01	1983-01-01
IF2000	10	0.50	1982-01-01	1983-01-01
IF2000	100	0.75	1982-01-01	1982-07-01
IF2000	110	0.50	1982-03-01	1982-07-01
IF2000	110	0.50	1982-10-01	1983-01-01
MA2100	10	0.50	1982-01-01	1982-11-01
MA2100	20	1.00	1982-01-01	1982-03-01
MA2110	10	1.00	1982-01-01	1983-02-01
MA2111	40	1.00	1982-01-01	1983-02-01
MA2111	50	1.00	1982-01-01	1092-06-01
MA2111	60	1.00	1982-06-01	1983-02-01
MA2112	60	2.00	1982-01-01	1982-07-01
MA2112	70	1.50	1983-02-01	1983-02-01
⋮	⋮	⋮	⋮	⋮

Figure 128. Partial Contents of PROJ_ACT Table

Column Name	Description
PROJNO	Project number
ACTNO	Activity number
ACSTAFF	Estimated mean number of employees needed to staff the activity
ACSTDATE	Estimated activity start date
ACENDATE	Estimated activity completion date

Figure 129. Columns of the PROJ_ACT Table

The table has a composite primary key and was created with:

```
CREATE TABLE PROJ_ACT
     (PROJNO     CHAR(6)      NOT NULL,
      ACTNO      SMALLINT     NOT NULL,
      ACSTAFF    DECIMAL(5,2)         ,
      ACSTDATE   DATE         NOT NULL,
      ACENDATE   DATE                 ,
      PRIMARY KEY (PROJNO, ACTNO, ACTSTDATE),
      FOREIGN KEY R_PROJ2 (PROJNO) REFERENCES PROJECT
              ON DELETE RESTRICT,
      FOREIGN KEY R_ACTIV (ACTNO) REFERENCE ACTIVITY
              ON DELETE RESTRICT)
```

Relationship to Other Tables

PROJ_ACT is a parent of the EMP_ACT table.

It is a dependent of:

- The ACTIVITY table; the foreign key on ACTNO in the PROJ_ACT table references the primary key, ACTNO, in the ACTIVITY table.
- The PROJECT table; the foreign key on PROJNO in the PROJ_ACT table references the primary key, PROJNO, in the PROJECT table.

EMP_ACT Table

The EMP_ACT table identifies the employee performing each activity listed for each project. The table in Figure 130 on page 280 shows some of the rows in this table. Figure 131 on page 280 shows a description of the columns.

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000130	IF1000	90	1.00	1982-01-01	1982-10-01
000130	IF1000	100	.50	1982-10-01	1983-01-01
000140	IF1000	90	.50	1982-10-01	1983-01-01
000030	IF1000	10	.50	1982-06-01	1983-01-01
000030	IF2000	10	.50	1982-01-01	1983-01-01
000140	IF2000	100	1.00	1982-01-01	1982-03-01
000140	IF2000	100	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-10-01	1983-01-01
000010	MA2100	10	.50	1982-01-01	1982-11-01
000110	MA2100	20	1.00	1982-01-01	1982-03-01
000020	PL2100	30	1.00	1982-01-01	1982-09-15
000010	MA2110	10	1.00	1982-01-01	1983-02-01
000220	MA2111	40	1.00	1982-01-01	1983-02-01
⋮	⋮	⋮	⋮	⋮	⋮

Figure 130. Partial Contents of EMP_ACT Table

Column Name	Description
EMPNO	Employee number
PROJNO	Project number of the project to which the employee is assigned
ACTNO	Activity number within a project to which an employee is assigned
EMPTIME	A proportion of the employee's full time (between 0.00 and 1.00) to be spent on the project from EMSTDATE to EMENDATE
EMSTDATE	Date the activity starts
EMENDATE	Completion date of the activity

Figure 131. Columns of the EMP_ACT Table

Since the table has foreign keys referencing EMPLOYEE and PROJ_ACT, those tables must be created first.

This table was created with:

```
CREATE TABLE EMP_ACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       EMPTIME   DECIMAL(5,2)             ,
       EMSTDATE  DATE                     ,
       EMENDATE  DATE                     ,
       FOREIGN KEY R_PROACT (PROJNO,ACTNO,EMSTDATE)
               REFERENCES PROJ_ACT ON DELETE RESTRICT,
       FOREIGN KEY R_EMPLY3 (EMPNO) REFERENCES EMPLOYEE
               ON DELETE CASCADE          )
```

Relationship to Other Tables

The EMP_ACT table is a dependent of:

- The EMPLOYEE table; the foreign key on EMPNO in the EMP_ACT table references the primary key, EMPNO, in the EMPLOYEE table.
- The PROJ_ACT table; the foreign key on the set of PROJNO, ACTNO, EMSTDATE in the EMP_ACT table references the primary key, PROJNO, ACTNO, ACSTDATE, in the PROJ_ACT table.

IN_TRAY Table

The IN_TRAY table contains a person's note log. The table contents are shown in Figure 132; a description of the columns is shown in Figure 133.

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1965-01-01-07.00.00	SQLDBA	English	Here is a note from your DBA.

Figure 132. IN_TRAY Table Contents

Column Name	Description
RECEIVED	Date and time note was received
SOURCE	User id of person sending note
SUBJECT	Brief description
NOTE_TEXT	The text of the note

Figure 133. Columns of the IN_TRAY Table

This table was created with:

```
CREATE TABLE IN_TRAY
     (RECEIVED     TIMESTAMP   NOT NULL,
      SOURCE       CHAR(8)     NOT NULL,
      SUBJECT      CHAR(64)            ,
      NOTE_TEXT    VARCHAR(4000)       )
```

CL_SCHED Table

The CL_SCHED table describes a classroom schedule. The table contents are shown in Figure 134; a description of the columns is shown in Figure 135.

CLASS_CODE	DAY	STARTING	ENDING
101:KAR	2	14.10.00	16.10.00
202:LMM	3	14.40.00	16.40.00
303:RAR	4	09.00.00	09.40.00

Figure 134. CL_SCHED Table Contents

Column Name	Description
CLASS_CODE	Class Code (room:teacher)
DAY	Day number of four day schedule
STARTING	Class start time
ENDING	Class end time

Figure 135. Columns of the CL_SCHED Table

This table was created with:

```
CREATE TABLE CL_SCHED
     (CLASS_CODE      CHAR(7)  NOT NULL,
      DAY             SMALLINT NOT NULL,
      STARTING        TIME     NOT NULL,
      ENDING          TIME     NOT NULL)
```

Note: For more information about data types, refer to the *DB2 Server for VSE Application Programming* or the *DB2 Server for VM Application Programming* manuals.

Appendix C. Summary of ISQL PF Keys

Listed below is a summary of the default PF key functions:

PF1, PF13 Issues a HELP command, which retrieves an explanation of how to use HELP information and a list of the topics available.

PF2, PF14 Issues a START command, which starts the statement in the SQL command buffer (the current SQL statement).

PF3, PF15 Issues an END command.

PF4, PF16 Issues a PRINT command, which requests the currently displayed query result to be printed on the designated printer.

PF5, PF17 Issues a RECALL command, which displays the contents of the SQL command buffer.

PF6, PF18 Not assigned.

PF7, PF19 Issues a BACKWARD command, which moves your view of the query result backward one-half display.

PF8, PF20 Issues a FORWARD command, which moves your view of the query result forward one-half display.

PF9, PF21 Issues a HOLD command.

PF10, PF22 Issues a LEFT 1 command, which moves your view of the query result one column to the left.

PF11, PF23 Issues a RIGHT 1 command, which moves your view of the query result one column to the right.

PF12, PF24 Issues a RETRIEVE command, which displays the last input line from the SQL command buffer and places it in the input area. Each successive use of RETRIEVE retrieves an earlier input line. When there are no more lines in the SQL command buffer to be retrieved, the newest line (the last one entered in the buffer) is again retrieved.

Using PF Keys in CMS FULLSCREEN Mode (DB2 Server for VM)

If you are using ISQL in CMS FULLSCREEN mode, you may want to reset the functions of some PF keys. You can reset your keys by executing a routine. The routine can be a PROFILE routine which is executed automatically every time you start ISQL. To reset the PF key functions, you can include the following example commands in the routine:

```
CMS
SET LINEND OFF
SET CMSPF 07 BACKWARD NOECHO #WM SCROLL BACKWARD CMS 1
SET CMSPF 08 FORWARD NOECHO #WM SCROLL FORWARD CMS 1
SET LINEND ON
RETURN
```

This addition to your routine lets you use PF7 and PF8 in ISQL command mode to scroll forward and backward in a manner similar to the CMS FULLSCREEN mode.

For information about creating and running routines, see Chapter 9, “Creating and Using Routines” on page 125.

You can also reset the PF keys every time you use ISQL. For more information on resetting the PF keys, see “Routines to Which Parameters Can Be Passed (DB2 Server for VM)” on page 125.

Appendix D. Summary of SQL Statements for Interactive Use

For syntax diagrams, semantic descriptions, rules, and situations where you would use the following SQL statements, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

- ACQUIRE DBSPACE
- ALTER DBSPACE
- ALTER TABLE
- COMMENT ON
- COMMIT
- CONNECT
- CREATE INDEX
- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DELETE (searched DELETE only)
- DROP DBSPACE
- DROP INDEX
- DROP PROGRAM
- DROP SYNONYM
- DROP TABLE
- DROP VIEW
- EXPLAIN
- GRANT PROGRAM PRIVILEGES
- GRANT SYSTEM AUTHORITIES
- GRANT TABLE OR VIEW PRIVILEGES
- INSERT
- Interactive select (see SELECT in the *DB2 Server for VSE & VM SQL Reference* manual)
- LABEL ON
- LOCK DBSPACE
- LOCK TABLE
- REVOKE
- ROLLBACK
- UPDATE (searched UPDATE only)
- UPDATE STATISTICS

Appendix E. Suppressing the ISQL Sign-On Display for DB2 Server for VSE

In addition to typing **isql** at a CICS terminal, to start ISQL, you can type the following command at a CICS terminal. This command suppresses the ISQL signon display and related terminal messages.

```
►►──ISQL──#r_id#──┬─userid/password─────────────┬──────────────────────────────────►
                  └─userid/password/server_name─┘

►──┬───────────────────────────────────────┬───────────────────────────────────────►◄
   └─routine_name──┬────────────────────┬──┘
                   └─(parameter_list)───┘
```

Where:

r_id	Is a 1–4 character CICS transaction identifier or 1 to 4 blanks.
#	Stands for a hexadecimal byte X'FF' positioned immediately before and after r_id to mark the beginning and end of r_id.
userid/password	Is the ISQL signon user ID and password. You must type the slash (/). Since the *server_name* is not specified, the *userid/password* will be used to connect to the default server.
userid/password/server_name	Is the ISQL signon user ID, password, and the server-name. You must type the slash (/). The *userid/password* will be used to connect to the specified server.
routine_name	Is optional. Refer to "Using the ISQL Transaction Identifier (DB2 Server for VSE)" on page 126.
(*parameter_list*)	Is optional. Refer to "Using the ISQL Transaction Identifier (DB2 Server for VSE)" on page 126.

This method of invoking ISQL is primarily designed for VSE system programs:

- Enter into an ISQL session directly from another interactive session. The ISQL signon display is suppressed so that the user of the interactive session can enter an ISQL session without doing the formal ISQL signon steps. The user ID and password supplied in the command are processed as if they were supplied by the signon display.
- Return to the interactive session when the ISQL session ends, passing any ISQL ending message to the interactive session. Before ISQL is ended, it starts the CICS transaction identified by *r_id* (provided *r_id* is not all blank characters) using the CICS START command. Ending messages from ISQL are passed as data in the START command. ISQL returns to CICS if *r_id* is blank.

The hexadecimal byte X'FF' is usually not available with terminal keyboards. The sample program, as shown in Figure 136 on page 288, illustrates one method of displaying the command on the terminal and prompting the user to start ISQL.

```
         TITLE 'STARTING ISQL WITHOUT THE SIGN-ON SCREEN'
****************************************************************
* THIS PROGRAM WRITES 2 LINES TO THE USER TERMINAL (24X80):    *
* - LINE 1 IS FOR SETTING UP THE INVOCATION OF ISQL WITHOUT    *
*   DISPLAYING THE SIGN-ON SCREEN.                             *
* - LINE 2 PROMPTS THE USER TO EXECUTE LINE 1 WITH THE ENTER   *
*   KEY, OR QUIT WITH THE CLEAR KEY.                           *
****************************************************************
         PRINT GEN
DFHEISTG DSECT
INSTRUCT CSECT
         SPACE
         EXEC CICS SEND FROM(ISQLSTR) FLENGTH(SENDLEN) ERASE
         EXEC CICS RETURN
ISQLSTR  EQU   *
LINE1    DC    X'1140401D4D'   LINE 1: INVISIBLE, MDT ON,
*                                      UNPROTECTED.
         DC    C'ISQL'
         DC    X'FF'           DELIMITER BYTE
         DC    C'TRX0'         TRANS-ID TO BE INVOKED AT ISQL
*                                                        EXIT.
         DC    X'FF'           DELIMITER BYTE
         DC    C'SQLDBA/SQLDBAPW'      USER-ID/PASSWORD TO ISQL
*
LINE2    DC    X'11C1501D40'    LINE 2: VISIBLE, MDT OFF,
*                                       UNPROTECTED.
         DC    C'PRESS ENTER KEY TO INVOKE ISQL,'
         DC    C' OR CLEAR KEY TO QUIT'
STRLEN   EQU   *-ISQLSTR
SENDLEN  DC    A(STRLEN)
         LTORG
         END
```

Figure 136. Starting ISQL Without the Sign-on Display

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VM Application Programming*, SC09-2661
- *DB2 Server for VM Database Administration*, SC09-2654
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VM Diagnosis Guide and Reference*, LC09-2672
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VM Master Index and Glossary*, SC09-2666
- *DB2 Server for VM Messages and Codes*, GC09-2664
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE & VM Quick Reference*, SC09-2670
- *DB2 Server for VM System Administration*, SC09-2657

DB2 Data Spaces Support Publications

- *DB2 Server Data Spaces Support for VM/ESA*, SC09-2675

DB2 Server for VSE Publications

- *DB2 Server for VSE Application Programming*, SC09-2662
- *DB2 Server for VSE Database Administration*, SC09-2655
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VSE Diagnosis Guide and Reference*, LC09-2673
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE Installation*, GC09-2656
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VSE Master Index and Glossary*, SC09-2667
- *DB2 Server for VSE Messages and Codes*, GC09-2665
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE System Administration*, SC09-2658
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2669
- *DB2 Server for VSE & VM SQL Reference*, SC09-2671

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2677
- *DRDA: Every Manager's Guide*, GC26-3195
- *IBM SQL Reference, Version 2, Volume 1*, SC26-8416
- *IBM SQL Reference*, SC26-8415

VM/ESA Publications

- *VM/ESA: General Information*, GC24-5745
- *VM/ESA: VMSES/E Introduction and Reference*, GC24-5837
- *VM/ESA: Installation Guide*, GC24-5836
- *VM/ESA: Service Guide*, GC24-5838
- *VM/ESA: Planning and Administration*, SC24-5750
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA: REXX/EXEC Migration Tool for VM/ESA*, GC24-5752
- *VM/ESA: Conversion Guide and Notebook*, GC24-5839
- *VM/ESA: Running Guest Operating Systems*, SC24-5755
- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: Group Control System*, SC24-5757
- *VM/ESA: System Operation*, SC24-5758
- *VM/ESA: Virtual Machine Operation*, SC24-5759
- *VM/ESA: CP Programming Services*, SC24-5760
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS Application Development Guide for Assembler*, SC24-5763
- *VM/ESA: CMS Application Development Reference for Assembler*, SC24-5764

- *VM/ESA: CMS Application Multitasking*, SC24-5766
- *VM/ESA: CP Command and Utility Reference*, SC24-5773
- *VM/ESA: CMS Primer*, SC24-5458
- *VM/ESA: CMS User's Guide*, SC24-5775
- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Pipelines User's Guide*, SC24-5777
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: XEDIT User's Guide*, SC24-5779
- *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780
- *VM/ESA: Master Index and Glossary*, SC09-2398
- *VM/ESA: Quick Reference*, SX24-5290
- *VM/ESA: Performance*, SC24-5782
- *VM/ESA: Dump Viewing Facility*, GC24-5853
- *VM/ESA: System Messages and Codes*, GC24-5841
- *VM/ESA: Diagnosis Guide*, GC24-5854
- *VM/ESA: CP Diagnosis Reference*, SC24-5855
- *VM/ESA: CP Diagnosis Reference Summary*, SX24-5292
- *VM/ESA: CMS Diagnosis Reference*, SC24-5857
- *VM/ESA: CMS Data Areas and Control Blocks*, SC24-5858
- *VM/ESA: CP Data Areas and Control Blocks*, SC24-5856
- *IBM VM/ESA: CP Exit Customization*, SC24-5672
- *VM/ESA REXX/VM User's Guide*, SC24-5465
- *VM/ESA REXX/VM Reference*, SC24-5770

C for VM/ESA Publications

- *IBM C for VM/ESA Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA Language Reference*, SC09-2153
- *IBM C for VM/ESA Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM C for VM/ESA Programming Guide*, SC09-2151
- *IBM C for VM/ESA User's Guide*, SC09-2152

Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA) Publications

- *IBM VSE/ESA Administration*, SC33-6505
- *IBM VSE/ESA Diagnosis Tools*, SC33-6514
- *IBM VSE/ESA General Information*, GC33-6501
- *IBM VSE/ESA Guide for Solving Problems*, SC33-6510
- *IBM VSE/ESA Guide to System Functions*, SC33-6511
- *IBM VSE/ESA Installation*, SC33-6504
- *IBM VSE/ESA Messages & Codes*, SC33-6507
- *IBM VSE/ESA Networking Support*, SC33-6508
- *IBM VSE/ESA Operation*, SC33-6506
- *IBM VSE/ESA Planning*, SC33-6503
- *IBM VSE/ESA System Control Statements*, SC33-6513
- *IBM VSE/ESA System Macros User's Guide*, SC33-6515
- *IBM VSE/ESA System Macros Reference*, SC33-6516
- *IBM VSE/ESA System Utilities*, SC33-6517
- *IBM VSE/ESA Unattended Node Support*, SC33-6512
- *IBM VSE/ESA Using IBM Workstations*, SC33-6509

CICS/VSE Publications

- *CICS/VSE Application Programming Reference*, SC33-0713
- *CICS/VSE Application Programming Guide*, SC33-0712
- *CICS Application Programming Primer (VS COBOL II)*, SC33-0674
- *CICS/VSE CICS-Supplied Transactions*, SC33-0710
- *CICS/VSE Customization Guide*, SC33-0707
- *CICS/VSE Facilities and Planning Guide*, SC33-0718
- *CICS/VSE Intercommunication Guide*, SC33-0701
- *CICS/VSE Performance Guide*, SC33-0703
- *CICS/VSE Problem Determination Guide*, SC33-0716
- *CICS/VSE Recovery and Restart Guide*, SC33-0702
- *CICS/VSE Release Guide*, GC33-0700
- *CICS/VSE Report Controller User's Guide*, SC33-0705
- *CICS/VSE Resource Definition (Macro)*, SC33-0709
- *CICS/VSE Resource Definition (Online)*, SC33-0708
- *CICS/VSE System Definition and Operations Guide*, SC33-0706
- *CICS/VSE System Programming Reference*, SC33-0711
- *CICS/VSE User's Handbook*, SX33-6079
- *CICS/VSE XRF Guide*, SC33-0704

CICS/ESA Publications

- *CICS/ESA General Information*, GC33-0155

VSE/Virtual Storage Access Method (VSE/VSAM) Publications

- *VSE/VSAM Commands and Macros*, SC33-6532
- *VSE/VSAM Introduction*, GC33-6531
- *VSE/VSAM Messages and Codes*, SC24-5146
- *VSE/VSAM Programmer's Reference*, SC33-6535

VSE/Interactive Computing and Control Facility (VSE/ICCF) Publications

- *VSE/ICCF Administration and Operation*, SC33-6562
- *VSE/ICCF Primer*, SC33-6561
- *VSE/ICCF User's Guide*, SC33-6563

VSE/POWER Publications

- *VSE/POWER Administration and Operation*, SC33-6571
- *VSE/POWER Application Programming*, SC33-6574
- *VSE/POWER Installation and Operations Guide*, SH12-5329
- *VSE/POWER Networking*, SC33-6573
- *VSE/POWER Remote Job Entry*, SC33-6572

Distributed Relational Database Architecture (DRDA) Library

- *Application Programming Guide*, SC26-4773
- *Architecture Reference*, SC26-4651
- *Connectivity Guide*, SC26-4783
- *DRDA: Every Manager' s Guide*, GC26-3195
- *Planning for Distributed Relational Database*, SC26-4650
- *Problem Determination Guide*, SC26-4782

C/370 for VSE Publications

- *IBM C/370 General Information*, GC09-1386
- *IBM C/370 Programming Guide for VSE*, SC09-1399
- *IBM C/370 Installation and Customization Guide for VSE* GC09-1417
- *IBM C/370 Reference Summary for VSE*, SX09-1246
- *IBM C/370 Diagnosis Guide and Reference for VSE* LY09-1805

VSE/REXX Publication

- *VSE/REXX Reference*, SC33-6642

Other Distributed Data Publications

- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *VM/Directory Maintenance Licensed Program Operation and User Guide Release 4*, SC23-0437
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol*
- *SNA LU 6.2 Reference: Peer Protocols*
- *Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*
- *Distributed Data Management (DDM) List of Terms*
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

C/370 Publications

- *IBM C/370 Installation and Customization Guide*, GC09-1387
- *IBM C/370 Programming Guide*, SC09-1384

Communication Server for OS/2 Publications

- *Up and Running!*,GC31-8189
- *Network Administration and Subsystem Management Guide* SC31-8181
- *Command Reference*, SC31-8183
- *Message Reference*, SC31-8185
- *Problem Determination Guide*, SC31-8186

Distributed Database Connection Services (DDCS) Publications

- *DDCS User's Guide for Common Servers*, S20H-4793
- *DDCS for OS/2 Installation and Configuration Guide* S20H-4795

VTAM Publications

- *VTAM Messages and Codes*, SC31-6493
- *VTAM Network Implementation Guide*, SC31-6494
- *VTAM Operation*, SC31-6495
- *VTAM Programming*, SC31-6496
- *VTAM Programming for LU 6.2*, SC31-6497
- *VTAM Resource Definition Reference*, SC31-6498
- *VTAM Resource Definition Samples*, SC31-6499

CSP/AD and CSP/AE Publications

- *Developing Applications*, SH20-6435
- *CSP/AD and CSP/AE Installation Planning Guide*, GH20-6764
- *Administering CSP/AD and CSP/AE on VM*, SH20-6766
- *Administering CSP/AD and CSP/AE on VSE*, SH20-6767
- *CSP/AD and CSP/AE Planning*, SH20-6770
- *Cross System Product General Information*, GH23-0500

Query Management Facility (QMF) Publications

- *QMF General Information*, GC26-4713
- *QMF VSE/ESA Setup and Usage Guide*, GG24-4196
- *Managing QMF for VSE/ESA*, SC26-3252
- *Installing QMF on VSE/ESA*, SC26-3254
- *QMF Learner's Guide*, SC26-4714
- *QMF Advanced User's Guide*, SC26-4715
- *QMF Reference*, SC26-4716
- *Installing QMF on VM*, SC26-4718
- *QMF Application Development Guide*, SC26-4722
- *QMF Messages and Codes*, SC26-4834
- *Using QMF*, SC26-8078
- *Managing QMF for VM/ESA*, SC26-8219

DL/I DOS/VS Publications

- *DL/I DOS/VS Application Programming*, SH24-5009

COBOL Publications

- *VS COBOL II Migration Guide for VSE*, GC26-3150
- *VS COBOL II Migration Guide for MVS and CMS*, GC26-3151
- *VS COBOL II General Information*, GC26-4042
- *VS COBOL II Language Reference*, GC26-4047
- *VS COBOL II Application Programming Guide*, SC26-4045
- *VS COBOL II Application Programming Debugging*, SC26-4049
- *VS COBOL II Installation and Customization for CMS* SC26-4213
- *VS COBOL II Installation and Customization for VSE* SC26-4696
- *VS COBOL II Application Programming Guide for VSE* SC26-4697

Data Facility Storage Management Subsystem/VM (DFSMS/VM) Publications

- *DFSMS/VM User's Guide*, SC26-4705

Systems Network Architecture (SNA) Publications

- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA Format and Protocol Reference: Architecture Logic for LU Type 6.2*, SC30-3269
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Synch Point Services Architecture Reference* SC31-8134

Miscellaneous Publications

- *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100
- *Dictionary of Computing*, ZC20-1699
- *APL2 Programming: Using Structured Query Language*, SH21-1056
- *ESA/390 Principles of Operation*, SA22-7201

Related Feature Publications

- *Control Center Installation and Operations Guide for VM*, GC09-2678
- *Control Center Installation and Operations Guide for VSE*, GC09-2679
- *IBM Replication Guide and Reference*, S95H-0999

Index

Numerics

A

B

C

E

F

J

Q

R

Communicating Your Comments to IBM

DB2® Server for VSE & VM
Interactive SQL Guide and
Reference
Version 6 Release 1

Publication No. SC09-2674-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 416-448-6161
  - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
  - Internet: torrcf@ca.ibm.com
  - IBMLink: toribm(torrcf)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwt9)

Readers' Comments — We'd Like to Hear from You

DB2® Server for VSE & VM
Interactive SQL Guide and Reference
Version 6 Release 1

Publication No. SC09-2674-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SC09-2674-00

IBM®

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/4300-50
Program Number: 5648-A70

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-2674-00

Spine information:

DB2® Server for VSE & VM

Interactive SQL Guide and Reference

Version 6 Release 1