

SAS Publishing

SYSTEM 2000®
QUEST Language
and
System-Wide Commands

Version 12
First Edition

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 1991. *SYSTEM 2000® QUEST Language and System-Wide Commands, Version 12, First Edition*. Cary, NC: SAS Institute Inc.

SYSTEM 2000® QUEST Language and System-Wide Commands, Version 12, First Edition

Copyright © 1991, SAS Institute Inc., Cary, NC, USA

ISBN 1-55544-178-5

All rights reserved. Produced in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, August 1991

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Contents

List of Illustrations ix
Using This Book xi
Changes and Enhancements xvii

Usage

USING THE QUEST LANGUAGE

1 SYSTEM 2000 CONCEPTS AND TERMINOLOGY 1

Introduction 2
Using a SYSTEM 2000 Database 2
The Logical View of a SYSTEM 2000 Database 3
Creating a Database 10
The Physical View of a SYSTEM 2000 Database 15
Summary 21

2 QUALIFYING DATA 23

Introduction 24
Specifying Conditions 25
Constructing Expressions 26
Key Conditions and Non-Key Conditions 37

3 RETRIEVING DATA 39

Introduction 39
Retrieval Objects 40
Retrieval Commands 41
Format Options 48
Ordering Output 49

4 UPDATING DATA 53

Introduction 53
Commands for Updating Item Values 54
Commands for Updating Data Trees 61

5 SUPPLEMENTARY CAPABILITIES FOR RETRIEVING AND UPDATING 69

Controlling the Processing of Commands 70
Repeating Command Units 71
Accessing Derived Information 72
Stored Strings and Functions 75
Reloading a Database 80

6 GUIDELINES FOR USING THE QUEST LANGUAGE 81

Entering and Exiting the Software and the QUEST Processor 82
Issuing QUEST Commands 84
Security Considerations 85

Processing Environments 86
System-Wide Commands That Affect the QUEST Language 87
User Files 88
Messages 88

Reference

LOADING DATA

7 CHOOSING A METHOD FOR LOADING A DATABASE 7-1

Methods for Loading 7-1
Selecting a Method 7-3

8 USING THE LOAD COMMAND 8-1

Introduction 8-2
The LOAD Command 8-3
Preparing the Data File 8-10
Typical Problems with Loading 8-19
The Loading Process 8-24

RETRIEVING DATA

9 INTRODUCTION 9-1

The Retrieval-Clause 9-2
Retrieval Command Considerations 9-3
Retrieval Command Processing 9-4
Optimization Considerations for Retrieving 9-8
Inserting Comments 9-9

10 CREATING A COLLECT FILE: COLLECT 10-1

11 RETRIEVING IN COLUMNAR FORMAT: LIST 11-1

Layout Specifications 11-4
The S-Specification 11-25

12 RETRIEVING IN SEQUENTIAL FORMAT: PRINT 12-1

13 RETRIEVING KEY VALUE INFORMATION: TALLY 13-1

14 RETRIEVING VALUE STREAMS: UNLOAD 14-1

15 CHANGING OUTPUT FORMAT: FORMAT OPTIONS 15-1

Format Options 15-2
The FORMOP Command 15-22

16 SPECIFYING OUTPUT ORDER: ORDERING-CLAUSE 16-1

17 DISPLAYING VALUES TOGETHER: BY-CLAUSE 17-1

18 RETRIEVAL EXAMPLES BY TASK 18-1

UPDATING DATA

19 INTRODUCTION 19-1

Optimization Considerations for Updating 19-2

20 UPDATING DATA RECORDS 20-1

Introduction 20-2
Replacing a Null with a Value: ADD 20-6
Replacing Either a Value or a Null with a Value: ASSIGN 20-10
Replacing a Value with a Value: CHANGE 20-15
Replacing a Value with a Null: REMOVE 20-19

21 UPDATING DATA TREES 21-1

Introduction 21-2
Replacing One Data Tree with Another: ASSIGN TREE 21-4
Copying Data Trees: COPY TREE 21-10
Adding New Data Trees: INSERT TREE 21-15
Moving Data Trees: MOVE TREE 21-37
Removing Data Trees: REMOVE TREE 21-42

22 CREATING A LOGICAL UNIT OF WORK: FRAME AND END FRAME 22-1

23 UPDATE EXAMPLES BY TASK 23-1

QUALIFYING DATA

24 THE WHERE-CLAUSE 24-1

Introduction 24-2
Specifying Where-Clause Conditions 24-6
Specifying Expressions with Logical Operators: AND, OR, NOT 24-25
Designating Specific Records: HAS 24-29
Specifying a Position Among Siblings: AT 24-33
Processing Order of Operators 24-37

25 NORMALIZING WITH THE HAS OPERATOR 25-1

26 OPTIMIZATION OF WHERE-CLAUSES 26-1

Where-Clause Processing 26-2
Factors Affecting the Efficiency of Where-Clause Processing 26-5
Guidelines for Optimizing Where-Clauses 26-6

SUPPLEMENTARY CAPABILITIES

27 CHANGING THE DATE FORMAT: DATE FORMAT IS 27-1

28 DISPLAYING THE DATABASE DEFINITION: DESCRIBE 28-1

29 PADDING THE DISTINCT VALUES TABLE: ENABLE/DISABLE VALUES PADDING 29-1

30 OBTAINING THE CURRENT DATE AND TIME: SYSTEM STRINGS 30-1

31 CONTROLLING PROCESSING 31-1

Introduction 31-2
Controlling Output Volume: LIMIT/END LIMIT 31-2
Controlling Execution in Event of Errors: STOP/CONTINUE 31-15
Preventing Execution of Commands: ENABLE/DISABLE EXECUTION 31-18
Moving Updated Pages to Disk: CLEAR 31-19

32 REPEATING COMMAND UNITS: DITTO, PREVIOUS, SAME 32-1

Introduction 32-2
Repeating Action-Clauses: DITTO 32-2
Repeating Value Streams: PREVIOUS 32-4
Repeating Qualification Criteria: SAME 32-5
Specifying Which Where-Clause to Repeat: SAME IS DYNAMIC/STATIC 32-7
Multi-User Environment Considerations for Repetitive Processing 32-9

33 OBTAINING CALCULATIONS: AD HOC FUNCTIONS 33-1

34 OBTAINING ARITHMETIC STATISTICS: SYSTEM FUNCTIONS 34-1

Using System Functions 34-2
Specifying the Focal Record for a System Function 34-6

35 USING STORED STRINGS AND FUNCTIONS 35-1

Introduction 35-1
Invoking Stored Simple Strings 35-2
Invoking Stored Parametric Strings 35-6
Invoking Stored Simple Functions 35-10
Invoking Stored Parametric Functions 35-12

SYSTEM-WIDE COMMANDS

36 INTRODUCTION 36-1

37 CHANGING PROCESSORS 37-1

Using Processor Commands 37-1
Using Processor Lookup 37-3

38 ALLOCATING AND CHANGING THE USER FILES 38-1

Introduction to User Files 38-2
Dynamic Allocation of User Files 38-3
JCL for alternate user files 38-4
LOCAL Option for CMS and TSO 38-4
Changing Command Files: COMMAND FILE IS 38-5
Changing Data Files: DATA FILE IS 38-8
Changing Message Files: MESSAGE FILE IS 38-13
Changing Report Files: REPORT FILE IS 38-17
Examples of Using Alternate User Files 38-21

39 CHANGING THE ENTRY TERMINATOR WORD: ENTRY TERMINATOR IS . . 39-1

40 CHANGING THE SYSTEM SEPARATOR: SEPARATOR IS 40-1

41 CHANGING THE WHERE-CLAUSE DELIMITER: DELIMITER IS 41-1

42 CHANGING THE SYSTEM PROMPT: PROMPT IS 42-1

43 ECHOING COMMANDS: ECHO 43-1

44 INSERTING COMMENTS 44-1

45 MONITORING PERFORMANCE: TIMING 45-1

46 CONTROLLING EXECUTION OF COMMANDS: IF...THEN...ELSE 46-1

47 RECORDING IF...THEN...ELSE VARIABLES: SHOW 47-1

48 EXITING A SYSTEM 2000 SESSION: EXIT 48-1

APPENDICES

APPENDIX A: GUIDELINES FOR AVOIDING ERRORS A-1

Preventing Command Execution Errors A-1
Preventing Undesirable Results A-2

APPENDIX B: COLLATING SEQUENCE AND CHARACTER SET B-1

APPENDIX C: SPECIAL CHARACTERS C-1

APPENDIX D: RESERVED AND RESTRICTED WORDS AND CHARACTERS D-1

APPENDIX E: RESERVED FILE NAMES E-1

Index X-1

Illustrations

1.1 Logical Entry Containing Data Records 4
1.2 Relationship between Schema and Data Trees 5
1.3 Schema and Data Trees 6
1.4 Hierarchical Relationships in Schema and Data Trees 7
1.5 Path and Family 8
1.6 Subtrees 8
1.7 Disjoint and Related Data Records 9
1.8 DESCRIBE Output 13

7.1 Comparison of Methods to Load a Database 7-8

8.1 Loader Stream Format for a Logical Entry 8-11
8.2 Example of a Loader Stream 8-12
8.3 Loader Stream with Errors 8-19
8.4 Sample Load of File LOADER 8-22
8.5 Corrected Loader Stream for LOADER 8-24

10.1 Schemas for the ORDERS, PARTS, and VENDORS Databases 10-8
10.2 Records in ORDERS Database 10-9
10.3 Records in PARTS Database 10-10
10.4 Records in VENDORS Database 10-11

11.1 Title and Paging Specifications 11-7
11.2 Default Column Headings 11-11
11.3 Column Specifications 11-13
11.4 Actual Column Width for LIST Output 11-19
11.5 Default Column Widths for LIST Display 11-22

15.1 Effects of Format Options 15-3
15.2 Effects of the ZERO Format Option 15-16
15.3 Effects of the ZERO SUPPRESS Format Option 15-16

16.1 Values Used in Ordering-Clauses 16-2

20.1 Implied Data Record Numbering Used for Trace Notation 20-4

24.1 Key-Conditions and Non-Key-Conditions 24-24
24.2 Relative Position under a Parent 24-33

29.1 Adding a Value in the Distinct Values Table 29-3
29.2 Distinct Values Table Padding 29-4
29.3 Use of Distinct Values Table Padding 29-6

31.1 Immediate Results and Continuing Effects of CLEAR Commands 31-20

x

Using This Book

Purpose

This book provides both usage and reference information for the QUEST language of SYSTEM 2000 software, Version 12. SYSTEM 2000 software is SAS Institute's data management system for mainframe computer systems running under MVS and CMS operating systems.

This book is for SYSTEM 2000 users who retrieve and update databases, and it assumes you have Version 12 or later of SYSTEM 2000 software.

Organization of This Book

This book has three parts: Usage, Reference, and Appendices. The following sections describe the parts and list the chapters they contain.

Usage The Usage part discusses the organization of a SYSTEM 2000 database, SYSTEM 2000 terminology, retrieving data, updating data, and guidelines for using the QUEST language.

Chapter 1, "SYSTEM 2000 Concepts and Terminology"

Chapter 2, "Qualifying Data"

Chapter 3, "Retrieving Data"

Chapter 4, "Updating Data"

Chapter 5, "Supplementary Capabilities for Retrieving and Updating"

Chapter 6, "Guidelines for Using the QUEST Language"

Reference The Reference part describes the syntax of QUEST commands in detail and provides examples. It also describes the system-wide commands, which apply to all SYSTEM 2000 SCF languages.

Chapters 7 and 8 describe loading data.

Chapters 9 through 18 describe retrieving data.

Chapters 19 through 23 describe updating data.

Chapters 24 through 26 describe qualifying data.

Chapters 27 through 35 describe supplementary capabilities.

Chapters 36 through 48 describe the system-wide commands.

Appendices The appendices provide information on various special areas.

Appendix A, "Guidelines for Avoiding Errors"

Appendix B, "Collating Sequence and Character Set"

Appendix C, "Special Characters"

Appendix D, "Reserved Words and Characters"

Appendix E, "Reserved File Names"

Reference Aids

The following reference aids are located at the end of the book:

Index	identifies pages where specific topics and terms are discussed.
Foldout	provides the definition of the EMPLOYEE database and four sample logical entries. Most of the examples use the EMPLOYEE database.

Typographical Conventions

You will notice several type styles throughout the book. Their different purposes are summarized here.

roman	is the basic type style used for most text.
UPPERCASE ROMAN	is used for references in the text to SYSTEM 2000 command keywords and database component names.
italic	is used for terms that are defined in the text and to emphasize important information.
`MONOSPACE`	is used to show examples of commands. This book uses uppercase type for SYSTEM 2000 commands. You can enter your own commands in lowercase, uppercase, or a mixture of the two.

Syntax Conventions

SYSTEM 2000 command syntax uses the following syntax notation:

UPPERCASE ROMAN	indicates keywords. They must be spelled exactly as given in the syntax shown. Abbreviations for keywords are also shown in uppercase. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A keyword without brackets is a required keyword.

italic	indicates generic terms representing words or symbols that you supply. If the term is singular, supply one instance of the term. If the term is plural, you can supply a list of terms, separated by commas. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A term without brackets is a required term.
[ ]	enclose an optional portion of syntax. The brackets are not part of the command syntax. Multiple choices are stacked vertically with a bar to the left. Select only one.
| (vertical bar)	in syntax, means to select one of the vertically stacked choices. If the choices are in brackets, the choice is optional; if not, a choice is required. Then continue to the next portion of syntax, shown on the top line of the command syntax. In margins, indicates a technical change in the text for the latest version of the software.
. . . (ellipsis) . .	indicates that the previous syntax can be repeated. In examples, either vertical or horizontal ellipses indicates an omitted portion of output or a command.
b	indicates a significant blank in syntax or output. Generally, spaces indicate blanks, and the *b* is used only for emphasis.
* (asterisk)	is the default system separator throughout this book.
END	is the default entry terminator word throughout this book.

Note that symbols within syntax (such as parentheses, asterisks, or commas) are required unless enclosed in brackets or specifically noted as optional.

Example Conventions

The examples show you how to use the commands. You can run most of the examples using the EMPLOYEE database.

Examples are shown in uppercase. However, you can enter your own commands in lowercase, uppercase, or a mixture of both. By default, SYSTEM 2000 software translates keywords and component names entered in lowercase to all uppercase before processing, except when the data come from an alternate Command File or Data File. Due to this translation, you cannot have, for example, uppercase C3 referring to component number 3 and lowercase c3 referring to component name c3. Lowercase c3 will always be recognized as component number 3.

Page Numbering Conventions

Page numbering in the Usage part is different from that in the Reference part.

- Pages in the Usage part are numbered sequentially, that is, 1, 2, 3, and so on.
- Pages in the Reference part are numbered sequentially within the chapter number, that is, 8-1, 8-2, and so on.

Both page numbering conventions are reflected in the index. For example, the index entry for the MAP command might list page numbers 21 and 12-1. This means that usage information for the MAP command is located on page 21 of the Usage part and reference information starts in the Reference part on the first page of Chapter 12.

Additional Documentation

There are many SYSTEM 2000 publications available. To receive a free *Publications Catalog*, write to the following address:

SAS Institute Inc.
Book Sales Department
SAS Campus Drive
Cary, NC 27513

The books listed below should help you find answers to questions you may have about SYSTEM 2000 software.

- *SYSTEM 2000 Introductory Guide for SAS Software Users, Version 6, First Edition* (order #A55010) provides introductory information for SYSTEM 2000 software. It also explains how to use the SAS System with SYSTEM 2000 databases and provides examples.
- *SAS/ACCESS Interface to SYSTEM 2000 Data Management Software: Usage and Reference, Version 6, First Edition* (order #A56064) documents the ACCESS procedure, the DBLOAD procedure, and the QUEST procedure for the SAS/ACCESS interface to SYSTEM 2000 software. It explains how to create SAS/ACCESS descriptor files and how to use SAS programs with SYSTEM 2000 databases. Examples are provided.
- *SYSTEM 2000 Quick Reference Guide, Version 12, First Edition* (order #A55020) contains syntax, usage rules, and examples for all DEFINE, CONTROL, QUEST, and system-wide commands. Commands are organized alphabetically within each language, and the pages are color-coded for easy reference.
- *SYSTEM 2000 DEFINE Language, Version 12, First Edition* (order #A55012) serves as a usage and reference manual for the DEFINE language. This manual explains how to create and modify a SYSTEM 2000 database definition. It includes a detailed description and examples for each DEFINE command.
- *SYSTEM 2000 CONTROL Language, Version 12, First Edition* (order #A55013) describes how to use the CONTROL language to perform administrative tasks for SYSTEM 2000 databases, such as saving and restoring databases, assigning password security, specifying input and output files, and using the Coordinating Recovery feature.

- *SYSTEM 2000 REPORT Language, Version 12, First Edition* (order #A55014) explains how to use the REPORT language to produce reports from information contained in a SYSTEM 2000 database. This manual contains detailed reference material for using REPORT commands under MVS and CMS. Also included are examples of each command, of report composition, and of actual output.

- *SYSTEM 2000 PLEX Manual, Version 12, First Edition* (order #A55017) explains how to use the SYSTEM 2000 PLEX language. This manual details the use of PLEX commands and the methods of executing PLEX programs. It provides specific COBOL, PL/I, FORTRAN, and Assembler PLEX reference material and examples. The manual assumes a working knowledge of the programming language with which you will use SYSTEM 2000 software.

- *SYSTEM 2000 Messages and Codes, Version 12, First Edition* (order #A55015) contains all messages and codes issued by SYSTEM 2000 software. Probable causes and corrective actions accompany all error messages and warnings.

- *SYSTEM 2000 Product Support Manual, Version 12, First Edition* (order #A55016) describes how to configure SYSTEM 2000 software for single-user and Multi-User environments, how to specify SYSTEM 2000 files, and how to use user exits. Also included are descriptions of the executable modules, all execution parameters, the Accounting Log, and the Diagnostic Log.

- *SYSTEM 2000 CICS Interface, Release 11.5 under IBM OS* (order #A5511) provides supplemental details for using SYSTEM 2000 software under CICS at the macro level.

- *SYSTEM 2000 Interface to CICS (Command Level), Version 12, First Edition* (order #A55018) provides supplemental details for using SYSTEM 2000 software under CICS (command level). It includes system requirements, execution parameters, and available options.

- *SYSTEM 2000 CMS Supplement, Version 12, First Edition* (order #A55019) contains techniques, options, and commands that pertain to SYSTEM 2000 software under CMS.

- *SYSTEM 2000 DBMS Interactive Report Writing with Genius, Release 11.5 under IBM OS and CMS* (order #A5577) explains how to use the interactive Genius facility for producing reports and listings of data from SYSTEM 2000 databases. This manual contains detailed reference material and examples for all Genius prompts and user responses under MVS and CMS.

- *SYSTEM 2000 DBMS QueX User's Guide, Release 11.5A under IBM CICS, TSO, and CMS* (order #A5563) explains how to use the menu-driven QueX facility to enter data into and retrieve data from SYSTEM 2000 databases. This manual includes â step-by-step tutorial and reference information for QueX sessions.

- *SYSTEM 2000 DBMS QueX DBA Guide, Release 11.5A under IBM CICS, TSO, and CMS* (order #A5588) explains how to build user views for QueX sessions. This manual includes instructions for building, rebuilding, and deleting user views, for defining and deleting links in user views, and for maintaining the QueX system.

- *Technical Report: S2-106 Multi-User Tuning Tools for SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS* (order #A55001) describes the Tuning Tools feature for SYSTEM 2000 software. The Multi-User status console commands display details about thread, scratch pad, buffer, and queue usage. The Accounting and Diagnostic Log reports display information about system performance or about specific jobs and job types running in the system. Data are extracted from these logs into a SAS data set for use in reports, and they are summarized into historical SAS data sets for more general monthly, quarterly, or annual reports.

- *Technical Report: S2-107 XBUF Caching Feature in SYSTEM 2000 Software, Release 11.6 under IBM OS* (order #A55002) documents the Extended Buffer Manager (XBUF) feature, which provides several caching techniques for both single-user and Multi-User jobs. These techniques include simple "front-end" XBUF macros, with which you can take advantage of dual logging, statistics gathering, cache modeling, and caching of database files without having to employ the more complicated CACHE macro. Later, if you need to tailor caching for special applications, you can refer to the CACHE macro documentation also included in this report.

- *Technical Report: S2-110 QUEUE Language for SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS* (order #A55009) documents all aspects of the QUEUE language. Used together with the QUEST language manual, this technical report provides complete information on using the QUEUE facility in SYSTEM 2000 software.

- *SAS Technical Report S2-111, SYSTEM 2000 Internals: Database Tables, COMMON Blocks, and Debug Utilities, Version 12* (order #A55022) describes the structure of the internal tables in a SYSTEM 2000 database. The report details all table entries and the fields within the entries. Sample illustrations are included. In addition, the Update Log and Rollback Log formats are given. An Appendix lists all SYSTEM 2000 COMMON Data Areas, with an index and cross reference. This material is supplemental and is not required to use SYSTEM 2000 software.

- *SYSTEM 2000 Glossary and Subject Index, Version 12, First Edition* (order #A55021) defines specific terminology in SYSTEM 2000 software publications and includes a comprehensive index by subject to all Institute publications documenting SYSTEM 2000 software.

Changes and Enhancements

Introduction

This section lists the changes and enhancements made to the QUEST language of SYSTEM 2000 software for Version 12. This information is intended for users who have previous experience with the QUEST language.

IN and CTNSIN Operators

Two new where-clause operators, IN and CTNSIN, are available. These predicate operators imply two or more conditions joined with the OR operator. The IN operator implies an EQ condition, and the CTNSIN operator implies the CONTAINS operator. The IN and CTNSIN operators save time and avoid typing errors when you want to specify several similar conditions that have different values for the same schema item. For more information, see **Specifying the Predicate Operators IN and CTNSIN** on page 24-23.

Dynamic Allocation of User Files

You no longer have to allocate the files for the Command File, Data File, Message File, and Report File. SYSTEM 2000 software will allocate them dynamically if it does not find the S2KCOMD and S2KMSG DD statements in your JCL. Also, the S2KOUTP intermediate file can be dynamically allocated. For more information, see **Dynamic Allocation of User Files** on page 38-3.

Mixed Case Data

Version 12 accepts and recognizes keywords and component names entered in lowercase. By default, the software translates data to all uppercase before processing, except when the data come from an alternate Command File or Data File.

The one restriction with this enhancement is that all component names are uppercased by the software when you define the database. Prior to Version 12, uppercase C3 referred to component number 3 while lowercase c3 referred to component name c3. For example, it was possible for component number 1 to have a component name of c3. With Version 12, lowercase c3 will be uppercased by the software and therefore will always be recognized as component number 3.

To disable uppercase translation, you can set the execution parameter OPT043 to YES.

Ad Hoc Function Labels

Because Version 12 allows 10,000 components, SYSTEM 2000 software starts numbering ad hoc functions with 10001 when preparing the labels for output. In earlier releases the starting number was 5001.

USAGE

Using the QUEST Language

SYSTEM 2000® Concepts and Terminology

Qualifying Data

Retrieving Data

Updating Data

Supplementary Capabilities
for Retrieving and Updating

Guidelines for Using the QUEST Language

Chapter 1

SYSTEM 2000® Concepts and Terminology

INTRODUCTION 2

USING A SYSTEM 2000 DATABASE 2

THE LOGICAL VIEW OF A SYSTEM 2000 DATABASE 3
- *Logical Entries 3*
- *The Database Schema 4*
- *Relationships in a Schema Tree or Data Tree 7*
- *Missing Values 9*

CREATING A DATABASE 10
- *Defining a Database 10*
  - *Defining schema records 10*
  - *Defining schema items 10*
  - *Defining stored strings and functions 11*
- *Displaying a Database Definition 12*
- *Loading a Database 14*

THE PHYSICAL VIEW OF A SYSTEM 2000 DATABASE 15
- *The Index 15*
- *The Hierarchical Table and the Data Table 17*

SUMMARY 21

INTRODUCTION

The QUEST language consists of commands that allow you to retrieve and update data in a SYSTEM 2000 database.

The language includes easy-to-learn syntax and a complete set of diagnostic messages. When you issue a command, the software examines the command for syntax errors, and, if none are found, it executes the command. Retrieval commands produce immediate results, and update commands modify the data upon execution of the command, making the new data instantly available for retrieval.

The software also allows you to qualify a command so that it acts only on specific parts of the database. That is, you can retrieve all the data in a database or only specific parts.

For example, the following retrieval command displays the last names of all employees in the EMPLOYEE database:

```
PRINT LAST NAME:
```

However, when you qualify the data with a where-clause, this command displays the last names of only the employees in the Marketing Department:

```
PRINT LAST NAME WHERE DEPARTMENT EQ MARKETING:
```

Before you retrieve and update data, however, you must be familiar with some SYSTEM 2000 concepts and terms, which are provided in the following sections. Also, be sure that you have read Chapter 2 "SYSTEM 2000 Basics" in the *SYSTEM 2000 Introductory Guide for SAS Software Users, Version 6, First Edition,* which provides introductory information for new SYSTEM 2000 users.

USING A SYSTEM 2000 DATABASE

A SYSTEM 2000 database is an organized collection of related data designed to meet the needs of many users. For example, a database containing data about the employees of an organization can be used to satisfy the information requirements of both the personnel and payroll departments.

A SYSTEM 2000 database organizes related data according to a *hierarchical structure*; that is, you store and access data according to a specific rank or order. The ranking of data is based on one-to-many relationships among groups of associated data.

The software allows you to organize and access data in a manner suited to your needs without your having to be concerned with the physical storage of the data. In other words, the physical and logical structures are independent. However, the software uses both structures to store data in an efficient manner and to access data based on both the values and the relationships among the data.

THE LOGICAL VIEW OF A SYSTEM 2000 DATABASE

Logical Entries

A SYSTEM 2000 database consists of groups of logically related data called *logical entries.* Each logical entry contains sub-groups of related data called *data records.* Data records, in turn, contain *values.*

For example, in the EMPLOYEE database, logical entries contain data about individual employees. Each logical entry has a data record for personal data, such as the employee's last name, birthday, and so on. Each logical entry also has a data record pertaining to the position that the employee holds in the company, such as title, department, and pertinent dates. If the employee has held several positions, there is a data record for each position.

You can think of a database as a special kind of office file. For example, a personnel file can be compared to a database containing data about employees. A personnel file usually contains folders arranged in a specific order. Each folder contains a form with data on one employee. The form is usually divided into sections, one section for each set of related data. In the same manner, the EMPLOYEE database contains one logical entry for each employee, and the data found in the sections of the form correspond to the values found in the various data records.

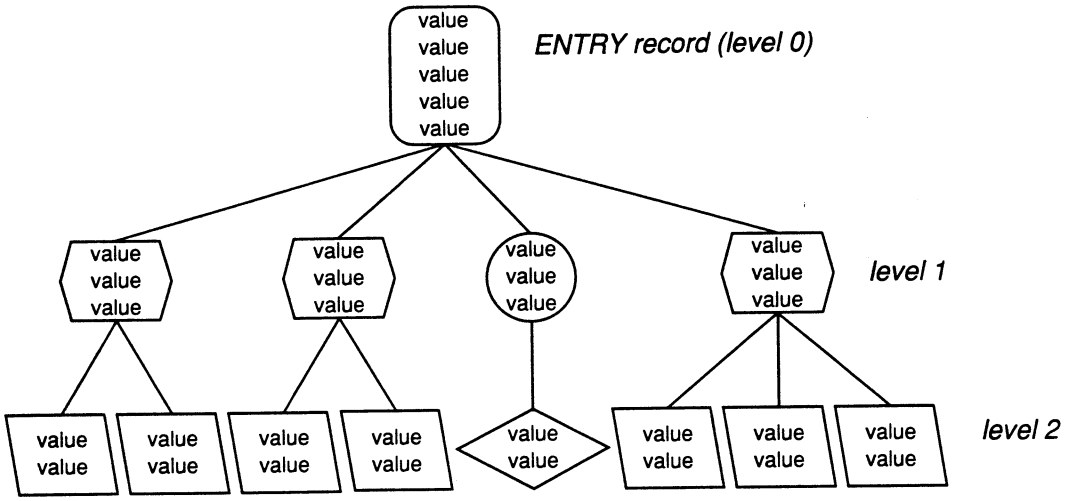
The relative positions of the sections within a form often imply relationships among the data in the various sections of the form. These relationships lend structure and meaning to the data contained in the form. In the same way, the relative positions of the data records in a logical entry reflect the hierarchical structure of the data. That is, each record is related to one record at the next higher level and to one or more records in the level immediately below.

Logical entries are represented by graphs called *data trees.* In a SYSTEM 2000 database, data trees are shown branching downward. The data record at the top (the root of the tree) is called the *ENTRY record.* That top level of of the database is designated as level 0, the next lower level is level 1, and so on.

Data records are represented in a data tree by geometric figures. Hierarchical relationships between data records are represented by lines connecting the geometric figures.

Illus. 1.1 on page 4 shows a data tree with a variety of data records. Note that a logical entry can have different kinds of records at the same level. This data tree has two kinds of records at level 1.

Illus. 1.1 Logical Entry Containing Data Records



The Database Schema

Continuing with the comparison between an office file and a database, the forms in each folder are copies of an original master. Each copy is completed with data pertinent to a specific employee. In the same way, logical entries in databases are filled-in copies of a master entry--the *database schema*.

Just as the blank form organizes information about an employee, the database schema is a blueprint for its data. The schema serves as a pattern by which you create logical entries for a database and interpret the data. A schema consists of *schema records* and *schema items*. A schema record is a set of schema items, and a schema item belongs to only one schema record.

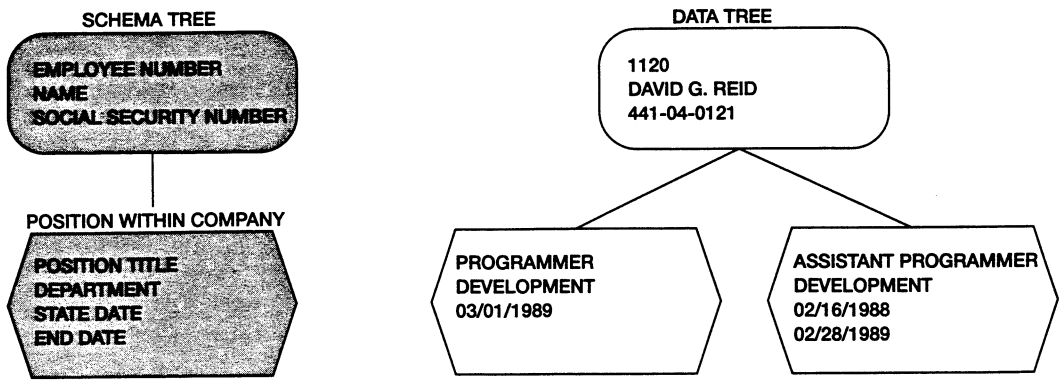
Each value in a data tree corresponds to a schema item. A schema item names and defines the characteristics or attributes of a group of values. For example, the schema item EMPLOYEE NUMBER (INTEGER 9999) refers to those integers that are used as employee identification numbers. The four 9s indicate that each employee number can contain up to four digits.

In the same way that data trees represent logical entries, the graphic representation of a database schema is referred to as a *schema tree*. A schema tree

- shows the different types of data records that a logical entry can have
- indicates the attributes that a value must fulfill in order to belong to a given type of data record
- specifies the relationship among records.

In Illus. 1.2 on page 5, the relationship between schema records and data records is shown by displaying a schema tree (the blueprint for the data) and a corresponding data tree (the specific data).

Illus. 1.2 Relationship between Schema and Data Trees



This illustration also shows that in a schema tree there is one distinct geometric figure for each schema record. But in a data tree, any geometric figure can occur once, more than once, or not at all. A schema record is defined only once; it can have as many corresponding data records as the data require.

Data records that are occurrences of the same schema record are said to be similar and are represented by the same geometric figure. Otherwise, they are dissimilar.

A database schema also allows you to define relationships between schema records. In a schema tree, these relationships are represented by lines connecting the various schema records. (Thus, in a data tree, two data records are joined by a line only if their corresponding schema records are also joined by a line in the schema tree.)

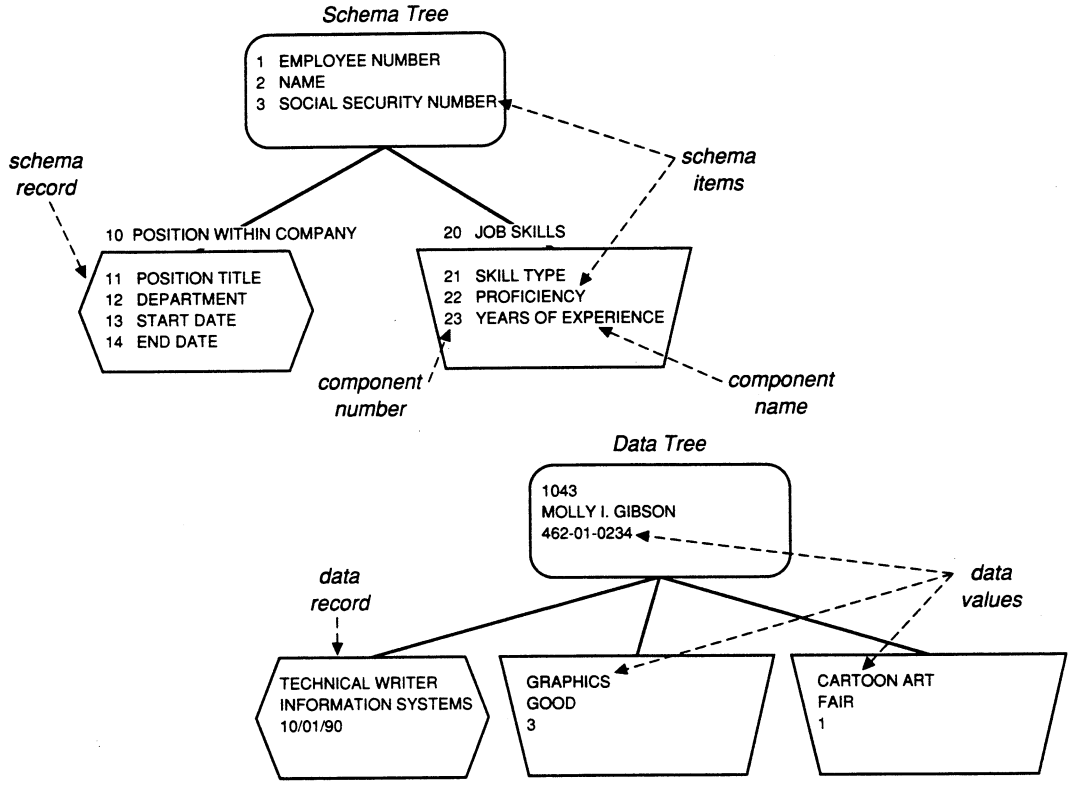
The schema items and schema records are the *schema components.* Each schema component is identified by both a *component number* and a *component name,* as shown below. (A component number can be referred to as a C-number, for example, C101.)

```
101* POSITION TITLE
 ↑          ↑
component  component
number     name
```

You can refer to a component by either its name or its number. Therefore, component numbers and names must be unique in a database schema to avoid ambiguity.

Illus. 1.3 on page 6 presents some of the terms previously introduced.

Illus. 1.3 Schema and Data Trees



Relationships in a Schema Tree or Data Tree

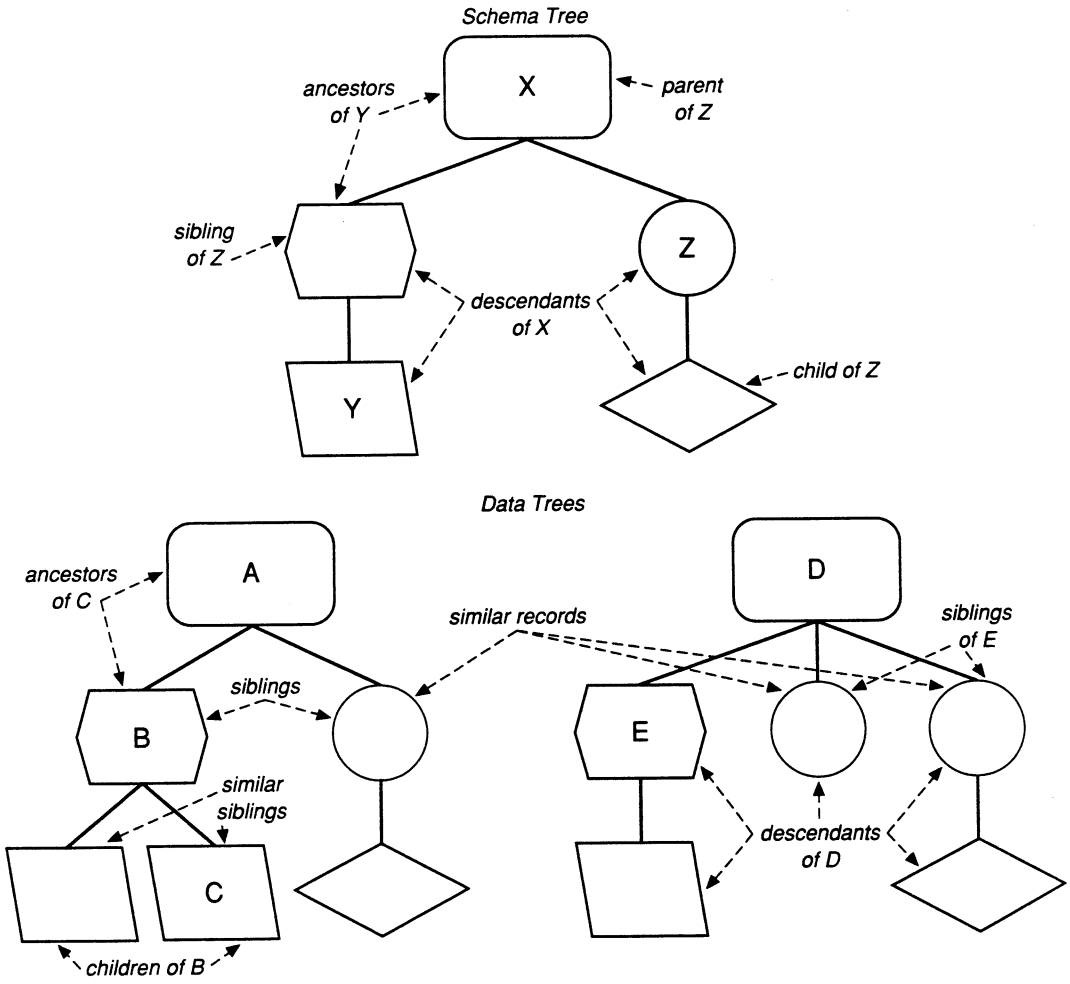
In either a schema tree or a data tree, relationships exist among the records. The terms used to refer to these relationships correspond in many respects to the terms used in a family tree.

The record immediately above a given record is its *parent*. A record can have only one parent, and the only orphan is the top record (the ENTRY record).

Record A is said to be an *ancestor* of record B if A is the parent of B, or if A is the parent of an ancestor of B. For example, the ENTRY record is an ancestor of all other records in its tree. A record is a *descendant* of all of its ancestors. The records immediately below a given record are its *children*. Records with the same parent are *siblings*.

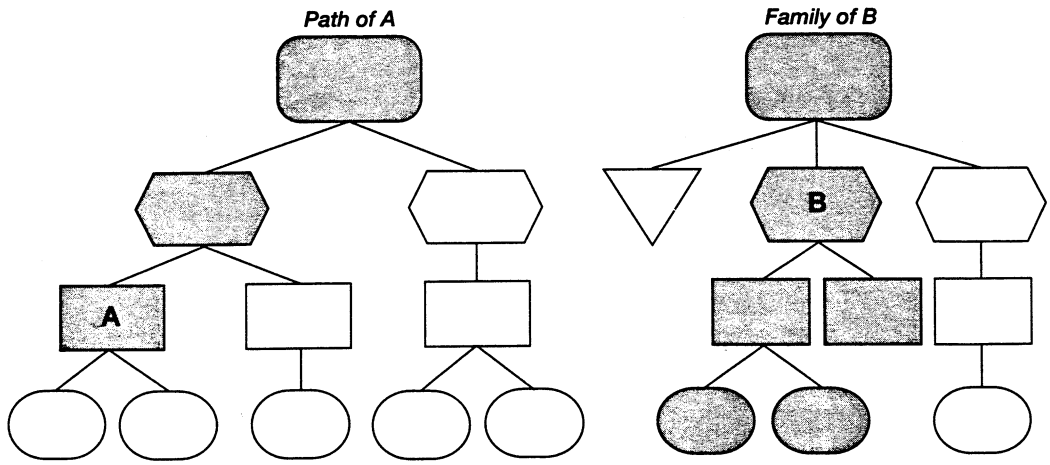
Other terms commonly used in connection with trees are *level*, *path*, and *family*. A *level* resembles a generation of people. The level number of a record is the same as the number of ancestors the record has. The level number of the schema items within a record is the same as the level number of the schema record.

Illus. 1.4 Hierarchical Relationships in Schema and Data Trees



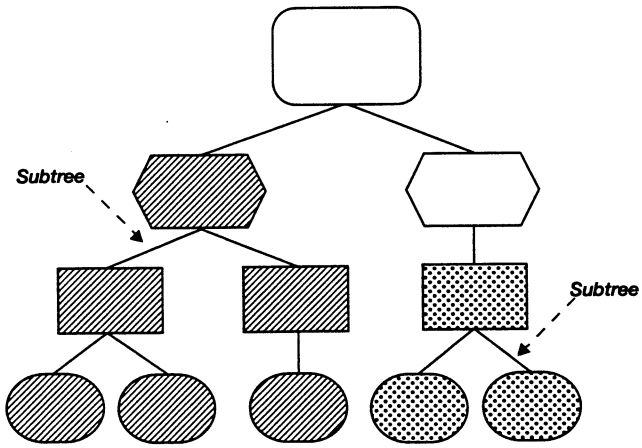
The *path* of a record is the record and all its *ancestors*. The following illustration highlights one path. A *family* consists of the record, all its ancestors, and all its descendants, as shown below.

Illus. 1.5 Path and Family



A *subtree* consists of a record and all its descendant records, as shown below.

Illus. 1.6 Subtrees

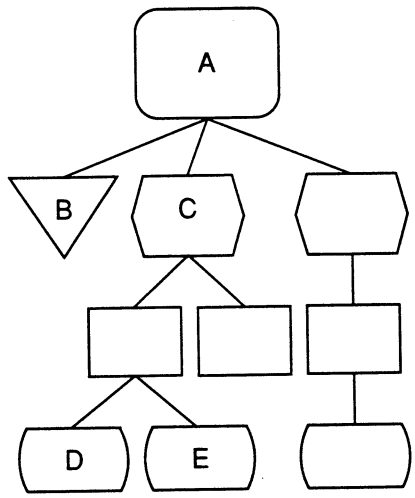


Schema records are *related* when they are members of the same path, that is, when one schema record is an ancestor of the other. Schema records that are not related are *disjoint*. That is, schema records are disjoint if their paths are different. However, data records are disjoint simply because they are occurrences of disjoint schema records, not because they themselves are on different paths.

In the following illustration

- Data records A, C, D, and E are related.
- Data records A and B are related.
- Data records D and E are related.
- Data records B and C are disjoint.

Illus. 1.7 Disjoint and Related Data Records



Missing Values

Just as a form does not need to be completely filled out to be useful, a logical entry does not need to contain a data record for each schema record, and each item in a given data record does not need to be valued.

Missing values are called *null items*. For example, see Logical Entry Three in the foldout at the end of this book. Component 13 there, is a null item, which means that Leonard Sly has not been assigned a security clearance.

A data record consisting entirely of nulls is called a *null record*. A null record can occur, for example, when there are data for a given data record but no data for its parent record. This parent record must be present in the database even though it is null since a parent record must exist for every record except the ENTRY record. The EMPLOYEE database has a null ENTRY record for one of the logical entries.

Sometimes a schema record contains no schema items. Such a schema record is called a *control record* or *control node*. Schema record 400 of the EMPLOYEE database is a control record; see the Database Schema in the foldout.

CREATING A DATABASE

Before you can use the QUEST language, a database must exist. That is, a database must be assigned a name and must be defined. After the database is defined, data can be loaded. This section provides an overview of how a database is defined and describes general ways of loading a database. It also describes how to display and interpret a database definition.

Defining a Database

The DEFINE language defines database components. You can also use the DEFINE language to modify a definition either before or after the database has been loaded. For detailed information about the DEFINE language, see *SYSTEM 2000 DEFINE Language, Version 12, First Edition*.

There are four types of components in a database definition: schema records, schema items, stored strings, and stored functions. The database schema consists of the schema records and schema items. Each component has a unique number and name assigned to it.

Defining schema records To define a schema record, specify a number and a name for the record and identify the component as a record. If the record is at level 2 or below, you must also specify a parent record.

Defining schema items To define a schema item, specify

1. a component number and a name, which labels the schema item
2. key or non-key status and padding, which determine whether values for the schema item are maintained in an index and whether the index has optional padding
3. the item type and picture, which determine how the data are used and displayed
4. the schema record to which the schema item belongs.

The software creates and maintains an index of values for each item defined as key. The software uses the index to quickly access records that are associated with a particular value (or values) of a key item.

Before you store values in a database, you must supply the software with information about how the values are to be displayed, stored, and used. The *item type* for a schema item identifies how the values for that item are to be stored and displayed. The manner of storage determines how the values can be used. For example, values consisting exclusively of digits can be stored in a manner suitable for computation. How the values are displayed and stored is also determined by the *picture* assigned to the item. For example, the picture assigned to an item that has decimal numbers as values indicates how many digits can be stored and where the decimal point is to appear when the values are displayed or used in computation.

The following item types are available:

- INTEGER items store integer values.
- DECIMAL items store decimal values.
- MONEY items also store decimal values, but they include a floating dollar sign at the left and CR at the right (if negative) when displayed.
- REAL (or FLOAT) items store full word (single-precision) floating point values.
- DOUBLE items store double word (double-precision) floating point values.
- DATE items store calendar dates in a fixed format, for example, 01/31/1990.
- CHARACTER items store textual data with trailing, leading, and extra internal blanks removed. For example, JOHN␣␣␣SMITH is stored and displayed as JOHN␣SMITH.
- TEXT items also store textual data; however, blanks are not removed. For example, ␣␣JOHN␣␣␣SMITH␣␣␣ is stored and displayed as ␣␣JOHN␣␣␣SMITH␣␣␣.
- UNDEFINED items store binary bit-string data containing any of the 256 EBCDIC characters.

An item's picture specifies how many characters a value can have. For INTEGER, DECIMAL, and MONEY items, the number of digits allowed is shown by using repetitions of the numeral 9, one 9 for each digit. For example, the item EMPLOYEE NUMBER has a picture of 9999. This means that no employee can have an employee number with more than four digits or greater than 9999. Dates and floating point values have no picture since internal storage is fixed.

However, you can control the format in which values are entered and displayed. The picture associated with CHARACTER and TEXT items indicates the number of places it takes to accommodate most of the values for an item. When a value exceeds its picture, the value is divided into two parts. The first part is kept in the same internal table as the other shorter values. The second part is stored in another table, called the Extended Field Table.

You can enter up to 250 characters for a CHARACTER or TEXT item. The picture is indicated by repetitions of the letter X. You can also use the letter X (or the numeral 9, in the case of numeric items) followed by the number of places in parentheses, for example, X(3) is equivalent to XXX and 9(4) is equivalent to 9999.

The picture associated with an UNDEFINED item is similar to the picture for CHARACTER and TEXT items. However, overflow is not allowed for UNDEFINED values; the number of characters in an UNDEFINED value must be equal to or less than the item's picture. As for CHARACTER and TEXT items, the picture is indicated by the repetitions of the letter X. You can also use the letter X followed by the number of places in parentheses, for example, UNDEFINED X(8).

Defining stored strings and functions Besides the schema records and items, a database definition can also include stored strings and functions as components. *Strings* allow you to store commands, parts of commands, or a series of commands. *Functions* allow you to store arithmetic expressions and to specify a numeric type for the display of output.

You define a string or function by specifying a component number and name, followed by the keyword STRING or FUNCTION, as appropriate, and the commands or parts of commands to be stored.

After strings and functions are stored in the definition, you can execute them by specifying their component name or number, which will in turn invoke the stored syntax.

Displaying a Database Definition

The DESCRIBE command displays all or part of a database definition. You can display individual components, a set of components, an entire schema, a record and its items, all stored strings, or all stored functions. If a Collect File exists, it too can be displayed. The DESCRIBE command honors all security measures that control access to components, displaying only those components available to the password in effect.

DESCRIBE RECORD displays the definition of a specified schema record and the schema items in it. For database components or Collect File items, you can specify the /DEFINE/ option to produce output acceptable as input to the DEFINE processor. In this manner, you can quickly create new database definitions that match existing definitions. The /AUTHORITIES/ option displays the access authorities for the password in effect. You can also combine the /DEFINE/ and /AUTHORITIES/ options (that is, /DEFINE,AUTHORITIES/) to produce DEFINE output that includes component access authorities for the current password.

A DESCRIBE listing begins with header information, as shown in Illus. 1.8 on page 13. (See the reference numbers that correspond to those below.)

1. The *system release number* is the release of SYSTEM 2000 software in use when the database was created, or a subsequent release if the database was converted from a prior release.

2. The *database name*, of course, is the name of the database.

3. The *definition number* is the number of times the MAP command has been issued to execute one or more DEFINE commands, that is, the number of times the definition has been modified.

4. The *database cycle number* is the number of times data were loaded or updated.

The schema records and items appear after the header information. The listing is organized by schema record. Items belonging to the level 0 record are printed first; however, the component name for C0 is listed only if it has been changed. Each line begins with the component number and name.

Schema records and their items are indented under their parent records. Items within a schema record are printed in the order in which they were defined. Items show the item type, picture, key/non-key status, record membership, and padding.

The database schema shown in the following illustration is an excerpt from the EMPLOYEE database. See the foldout at the end of this manual for the complete schema.

Illus. 1.8 DESCRIBE Output

```
PRINT/DOUBLE SPACE/:
DESCRIBE:

   SYSTEM RELEASE NUMBER  xx.x  1

   DATA BASE NAME IS      EMPLOYEE  2

   DEFINITION NUMBER             3  3

   DATA BASE CYCLE NUMBER        1  4

      1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
          ↑ component name
      2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
                                          ↑
      3*  FORENAME (NON-KEY CHAR X(20))       padding
                           ↑
      5*  BIRTHDAY (DATE) ← item type

     13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES )
     ↑ component number

    100*  POSITION WITHIN COMPANY (RECORD)   ← level 1 record

      101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
                                        ↑ picture
      106*  END DATE (NON-KEY DATE IN 100)

      110*  SALARY WITHIN POSITION (RECORD IN 100)
                                          ↑ record membership
        112*  PAY SCHEDULE (CHAR X(7) IN 110)

        114*  CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)

        120*  MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)

          123*  OVERTIME HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)

      130*  ADDITIONAL INFORMATION (RECORD IN 100)

        131*  LINE NUMBER (DECIMAL NUMBER 99.9 IN 130)

        132*  COMMENT TEXT (NON-KEY TEXT X(7) IN 130)
                              ↑ key status, not specified for key items
    200*  JOB SKILLS (RECORD)
    ↑ right sibling of record C100
      201*  SKILL TYPE (CHAR X(12) IN 200 WITH SOME FUTURE OCCURRENCES )

DESCRIBE FUNCTIONS: DESCRIBE STRINGS:

    1001*  AGE (INTEGER FUNCTION (((*FTODAY*-BIRTHDAY)/365.25)))

    501*  SKILL REPORT (STRING ( LIST LAST NAME WHERE C201 EQ *1* :))
```

Loading a Database

After you have defined a database, it is ready to be loaded. You can do this in several ways:

- Use PLEX. See the *SYSTEM 2000 PLEX Manual, Version 12, First Edition* for more information.
- Use the QUEST language LOAD command.
- Use the QUEST language INSERT TREE command.

Which method is better for an application depends on several factors, which are discussed in **Methods for Loading** on page 7-1. The principal decision is whether to use PLEX or the QUEST language.

To load data with a PLEX program, you must know one of the following programming languages: PL/I, COBOL, FORTRAN, or Assembler. The data must reside in a file (on disk or tape). Then, you can code a load program to load the data into the database.

Loading with the QUEST language does not require that you know a programming language. However, the data must be in a specific format called the *loader stream* format. If the data are already in machine readable form, they must be converted to loader stream format using a conversion program. If the data are not in machine readable form, you can create a loader stream directly from source documents in a logical entry format. No special coding is required, nor is it necessary to sort the data before loading.

Once a database is loaded, you can issue QUEST retrieval and update commands.

Thus far, this chapter has discussed the logical structure of a database, how a database is defined and loaded, and some of the terminology associated with these topics. *But how does this logical structure relate to the actual storage of data inside the software? What does SYSTEM 2000 software use to carry out this logical structuring?* The next section covers techniques the software uses to maintain a database in the logical structure you specified, which is independent of the physical storage order of the values.

You can use the software without any knowledge of the physical structure; however, some understanding of this structure can be helpful to make more efficient use of the available resources.

THE PHYSICAL VIEW OF A SYSTEM 2000 DATABASE

The previous sections have presented the concepts necessary for accessing a database. This section presents, in brief, the software's structure for physically storing your data, which consists of seven database tables.

1. The *Master Record* contains information about controlling the database (the database name, cycle, passwords, and so on) and date and time of creation and last update, along with Update Log information.

2. The *Definition Table* contains an internal representation of the database definition, including a description of each component.

3. The *Distinct Values Table* contains a multi-level index of the distinct values for each key item.

4. The *Multiple Occurrence Table* contains blocks of pointers to Hierarchical Table entries for key values that occur more than once for an item.

5. The *Hierarchical Table* contains the hierarchical structure of the database with parent, sibling, and descendant relationships for each data record in the Data Table.

6. The *Data Table* contains the actual fixed-length data records.

7. The *Extended Field Table* contains values exceeding the picture allotted for textual items, the overflow from component names, and the contents of stored strings and functions.

The discussion in this section focuses on four of the seven tables: the Distinct Values Table, the Multiple Occurrence Table, the Hierarchical Table, and the Data Table. The first two tables make up the index.

The Index

When you define a schema item, you specify whether the software is to create an index of values for that item. Each key item is indexed.

In a traditional personnel file, folders are usually arranged either alphabetically by employee name or numerically by employee identification number. When you need to access any piece of employee data other than the name (or number), for instance, programming languages an employee may know, you must examine each folder.

With an indexing system, the folders are still kept in a certain order, but separate indexes keep track of which folders contain certain information. In this system you do not need to go through every folder to find a certain piece of information; you consult the index instead.

For a SYSTEM 2000 database, you specify which items are to be indexed, and the software uses the index to access the appropriate data records for you. An item for which an index is created is a key item, implying that it provides easy access to the records containing its values. The values for non-key items are not indexed, although they can be searched sequentially.

You specify the key status of each item when you define your database, by using the keywords KEY or NON-KEY. If you need to create an index for a non-key item at a later time, you can use the DEFINE language to change the item to a key item. (See *SYSTEM 2000 DEFINE Language, Version 12, First Edition* for more information on modifying a database schema.) However, this method causes restructuring of the database. The CONTROL language CREATE INDEX command allows you to create an index of values for one or more non-key items without causing a restructuring. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information on the CREATE INDEX command.)

Reserving physically contiguous index space, called *padding*, for later additions to the index can make maintenance and use of the index more efficient.

For example, suppose ZIP CODE is a key item. The index for ZIP CODE without padding might appear as shown below:

ZIP CODE						
Values	Data Sheet Number					
78610	3	7				
78620	2	4	6	9	11	
78623	1	10	14			
78625	5	8	10	12		

Without padding, the addition of another employee whose zip code is 78620 would cause a new column to be added to the index. To provide for additional columns, you can specify padding for an item at the time it is defined.

On the other hand, if a new employee had a zip code of 78630, a new row would have to be added to the index. To allow for additional row space, you can issue the QUEST language ENABLE VALUES PADDING command, which creates blank rows.

In summary, you can request two types of padding, as shown below.

Values	Data Sheet Number					
78610	3	7				This area is padding for multiple occurrences specified in the schema item definition.
78620	2	4	6	9	11	
78623	1	10	14			
78625	5	8	10	12		
This area is padding for distinct values specified with ENABLE VALUES PADDING.						

You specify padding for multiple occurrences of values when you define the schema item. The example below shows one key item (SOCIAL SECURITY NUMBER) that does not need padding and another key item (GENDER) that does.

SOCIAL SECURITY NUMBER	
Values	Data Sheet Number
031-78-5430	2
102-78-8765	6
105-32-9011	9
109-07-5098	1
123-12-0987	8
170-83-7516	4
181-49-4592	10
210-65-2786	7
211-95-9608	3
234-67-8901	5

GENDER							
Values	Data Sheet Number						
FEMALE	1	5	9	10			
MALE	2	3	4	6	7	8	

Since each employee has a unique social security number, each value for SOCIAL SECURITY NUMBER can occur only once, so padding is not needed. On the other hand, the values for GENDER occur many times, so the schema item for GENDER includes the words WITH MANY FUTURE OCCURRENCES.

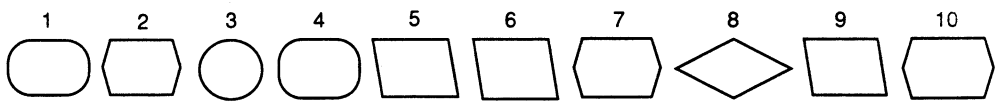
These examples give the impression that the index points to a "data sheet number" where a value can be found. Actually, the index points to entries in the Hierarchical Table. These entries correspond one-to-one with entries in the Data Table where a data record containing a given value is found.

The Hierarchical Table and the Data Table

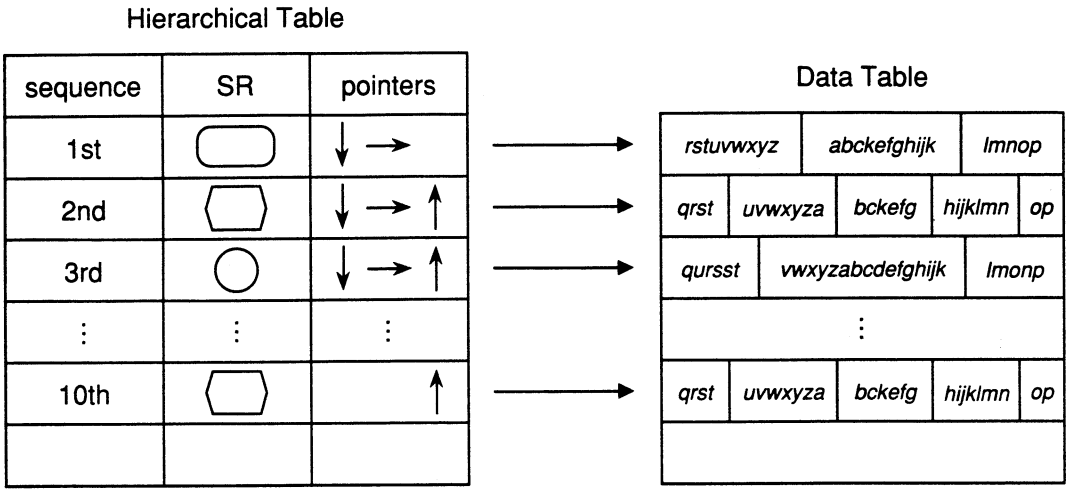
In order to do retrievals, the software must know where each data record is stored and how the data records are related to one another. The software maintains and uses two of the seven database tables for this purpose: the Hierarchical Table and the Data Table.

The Hierarchical Table contains data record pointers that indicate the logical relationships among data records in the database. The Data Table holds the actual values of all the data records. The order in which sets of pointers for data records are stored in the Hierarchical Table is not the logical order of the data, but rather the physical order of data storage.

Suppose various data records appear in physical storage sequence as shown below:



The Hierarchical Table and the Data Table are physically assembled as follows. (SR stands for schema record.)



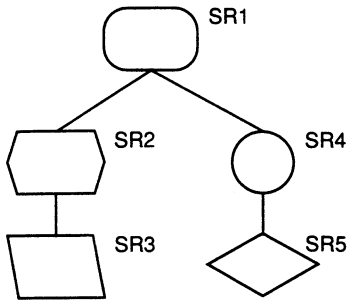
The Hierarchical Table contains a set of pointers for each data record, showing its place in the logical structure. The pointers for each data record point to other data records, that is, its first descendant, its next sibling, and its immediate ancestor, in logical order.

When a value is removed from a database by either the REMOVE or ASSIGN commands, the space in the Data Table where the value was stored is filled with binary zeroes. There is no change in the Hierarchical Table, even if all values in a data record are removed with those two commands. However, if a data record is removed from the database using either the REMOVE TREE or ASSIGN TREE commands, the Data Table is not modified. But the Hierarchical Table entry for that record is modified to indicate that the Data Table space for the removed record can be reused by another record. This space in the Data Table made available for new data records is called *reusable space*.

The reusable space must be used by a record of the same type as the record removed. For example, if a C100 record is removed, only a new C100 data record can be stored in that location in the Data Table.

The software keeps track of reusable space in the Data Table and the Hierarchical Table for each type of record. When maintaining reusable space, the software assigns the space freed most recently to the first new data record, regardless of the logical position of the new record with respect to the removed one. (In a Multi-User environment, space that is freed by a REMOVE TREE or ASSIGN TREE command is not immediately reusable if more than one user has the database open. Multi-User and single-user environments are discussed later.)

How do the Hierarchical Table pointers relate to the database schema? A schema tree could be defined as shown below:

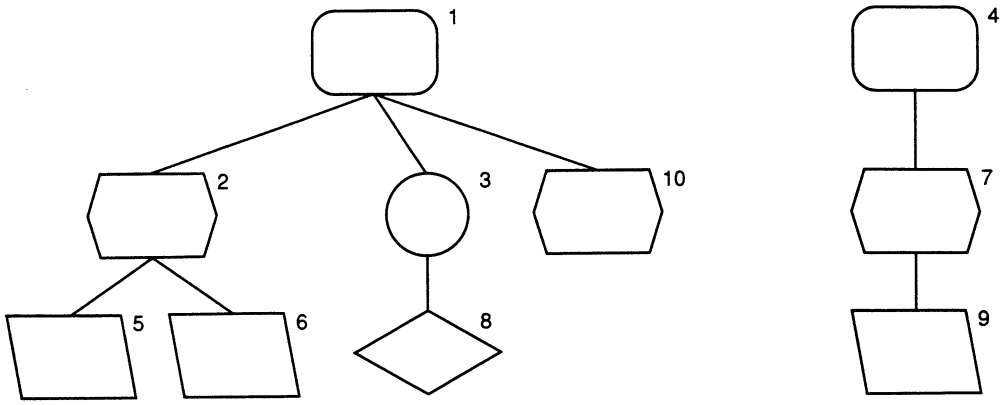


The ten data records shown in **The Hierarchical Table and the Data Table** on page 17 could have been entered in this database, and the resulting Hierarchical Table pointers would look like those in the following illustration. Notice that the fourth entry, which is for a level 0 record, contains an immediate ancestor pointer to the previous level 0 record.

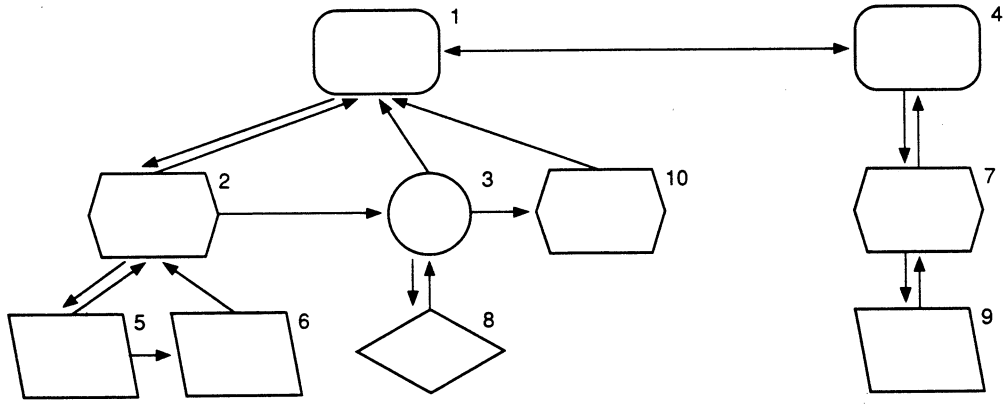
SR and sequence	POINTERS
1	↓2 →4
2	↓5 →3 ↑1
3	↓8 →10 ↑1
4	↓7 ↑1
5	→6 ↑2
6	↑2
7	↓9 ↑4
8	↑3
9	↑7
10	↑1

↓ pointer to first descendant
→ pointer to next sibling
↑ pointer to immediate ancestor

The corresponding logical structure of the data is shown by these data trees:

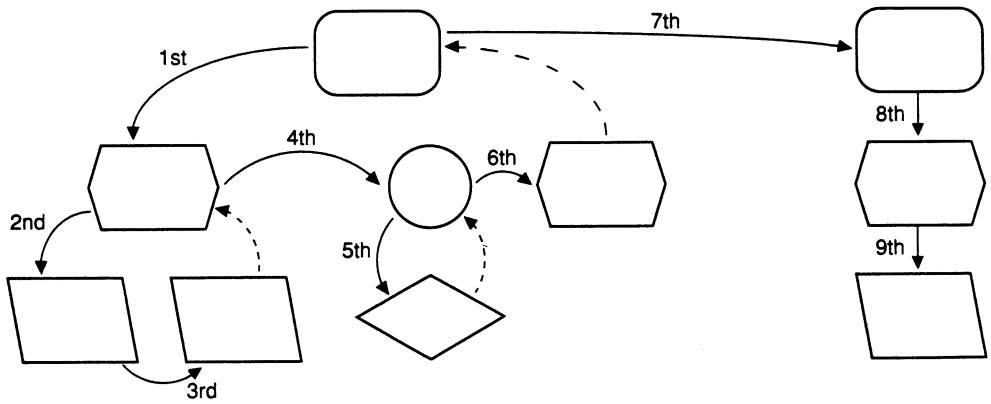


The Hierarchical Table pointers for the data trees are as follows:



The physical order of data storage cannot be the same as the logical order among data records. The order that you are usually concerned with when accessing any organized data is the logical order of those data.

How does the software move in logical order through the data trees? Generally speaking, movement is top-to-bottom and left-to-right through data tree structures. This traversal is shown below:



SUMMARY

This chapter introduced the logical and physical structures of a database. You may want to go over the chapter again, this time examining the four logical entries for the EMPLOYEE database provided in the foldout at the end of this manual.

This exercise enables you to become better acquainted with the database EMPLOYEE, which is used in most of the examples for this manual. The bold numbers appearing next to each data record in the four logical entries correspond to the physical storage sequence of the records in the Hierarchical Table and the Data Table.

Chapter 2

Qualifying Data

INTRODUCTION 24

SPECIFYING CONDITIONS 25
- *Operators 25*
  - *Unary operators 25*
  - *Binary operators 25*
  - *Ternary operators 25*
  - *Text search operators 25*
- *Processing Conditions 26*

CONSTRUCTING EXPRESSIONS 26
- *Logical Operator AND 26*
- *Logical Operator OR 27*
- *Logical Operator NOT 27*
- *AT Operator 29*
- *HAS Operator 32*
  - *The Importance of Focal records 34*
- *The Order of Processing Conditions 36*
- *Controlling the Processing Order with Parentheses 36*

KEY CONDITIONS AND NON-KEY CONDITIONS 37
- *Placement of Conditions 37*

INTRODUCTION

A database (or an office file) would be of little use if every logical entry (or folder) had to be accessed and read completely when data are needed from only some of entries (or folders). For example, a clerk needing information about the employees in the Marketing Department would first collect the folders of all Marketing employees and then obtain the needed information from each folder.

Most QUEST language retrieval and update commands give you the option of identifying those parts of the database that are relevant to the query or update. In processing these commands, the software follows a procedure similar to the clerk's.

1. First, it finds the relevant data trees.

2. Then it either retrieves data from those data trees or modifies the values.

In general, retrieval and update commands consist of an *action-clause*, in which you specify the data to be retrieved or the records to be updated, and an optional *where-clause*, in which you specify where the action is to take place.

> **An action-clause is the portion of a command to the left of the where-clause. If there is no where-clause, the action-clause is the entire command. For retrieval commands, the action-clause consists of the retrieval-clause and an optional ordering-clause.**

If you do not include a where-clause, the software assumes that the action affects the entire database and that all entries contain data to be accessed. For example, the following command causes the software to examine each entry and to extract from each one the value for LAST NAME:

```
PRINT LAST NAME:
```

When you include a where-clause in a retrieval or update command, you are determining what data trees the software will examine. A where-clause consists of one or more *conditions*. The software acts upon only those data trees containing data records whose values meet the specified conditions.

For example, the following command prints the last name of only those employees in the Marketing Department:

```
PRINT LAST NAME WHERE DEPARTMENT EQ MARKETING:
```

The software first makes a list of the C100 records that have the value MARKETING for the item DEPARTMENT. Then the software obtains the value for LAST NAME from the parent record of each of the C100 records in the list.

SPECIFYING CONDITIONS

A where-clause consists of one or more conditions, which are simple statements about items. Keywords called *operators* specify criteria that values must meet in order for the records containing those values to qualify for further processing.

Operators

You state conditions by using four types of operators:

- unary operators
- binary operators
- ternary operators
- text search operator.

Unary operators The unary operators are EXISTS and FAILS. They verify the existence or nonexistence of values. For example, the following command displays the last names of only those employees whose EDUCATION records contain a value for the item MINOR FIELD:

```
PRINT LAST NAME WHERE MINOR FIELD EXISTS:
```

Binary operators The binary operators are EQ, NE, GE, GT, LE, and LT. They compare an item with a value or with another item. For example, the following command displays the last names of only those employees whose ENTRY record contains the value 78741 for the item ZIP CODE:

```
PRINT LAST NAME WHERE ZIP CODE EQ 78741:
```

The following command displays the last name of each employee and the department where the person started working when hired:

```
PRINT LAST NAME, DEPARTMENT WHERE HIRE DATE* EQ START DATE*:
```

Ternary operators The ternary operators are SPANS, EQ, and NE. They compare an item with a range of values. For example, the following command displays the last names of those employees who were born in January, 1961:

```
PRINT LAST NAME WHERE BIRTHDAY SPANS 01/01/1961 * 01/31/1961:
```

Text search operators The text search operator is CONTAINS, and it searches for a set of characters within the values of an item. For example, this command displays the last names of those employees who live on Rim Rock Road or Rim Rock Street or Rim Rock Drive.

```
PRINT LAST NAME WHERE STREET ADDRESS CONTAINS RIM ROCK:
```

Processing Conditions

As the software processes a condition, it forms a list of records that contain the values satisfying the condition. These records are called *qualified data records*. For example, the qualified data records that result from processing the condition ZIP CODE EQ 78741 are those ENTRY records that contain the value 78741 for ZIP CODE.

These examples might give the impression that you can use only one simple condition in a where-clause. But many applications require more complex selection criteria. In those cases, your where-clause needs expressions.

CONSTRUCTING EXPRESSIONS

An *expression* consists of

- a single condition preceded by the NOT or the HAS operator
- a single condition followed by the AT operator
- two or more conditions joined by the AND or OR operator.

These operators act on the results (lists) produced by processing the conditions, and they derive new lists of qualified data records.

After the software processes the where-clause, there is one list of qualified data records.

- If the where-clause consists of only one condition, the list of qualified data records becomes the qualified data records for the where-clause.
- However, if the where-clause consists of a more complex expression, the list of qualified data records finally results from the consecutive processing of various lists of qualified records as each list is acted upon by the operators.

To be more specific, the lists contain Hierarchical Table addresses of the data records, not the data records themselves.

Logical Operator AND

When the logical operator AND joins two conditions, the resulting list of qualified data records consists of those records that meet both conditions.

Processing of each condition generates a list of data records arranged in numerical order by Hierarchical Table address. AND operates on the two lists by finding those numbers that are on both lists. This new list, also arranged in numerical order, contains the Hierarchical Table addresses of the data records qualified by the expression.

The following where-clause would look for employees who have four years of experience in COBOL:

```
WHERE SKILL TYPE EQ COBOL AND YEARS OF EXPERIENCE EQ 4
```

Logical Operator OR

When the logical operator OR joins two conditions, the resulting list of qualified records consists of those records that meet either or both conditions.

Processing of each condition generates a list of data records arranged in numerical order by Hierarchical Table address. OR operates on the two lists by combining both lists and discarding duplicate addresses. This new list, also arranged in numerical order, contains the Hierarchical Table addresses of the data records qualified by the expression.

The following where-clause would look for employees who either know COBOL or have four years of experience in any kind of skill:

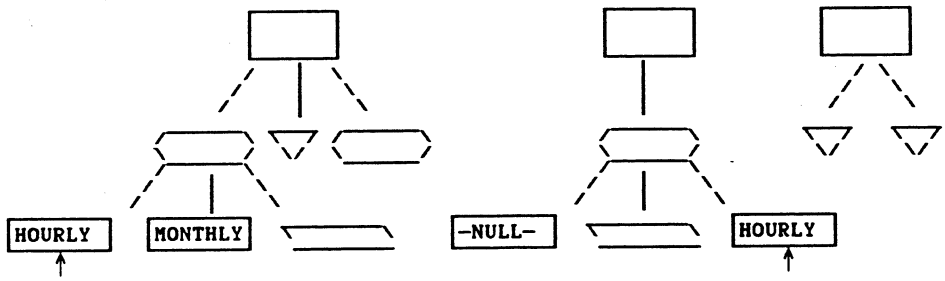
```
WHERE SKILL TYPE EQ COBOL OR YEARS OF EXPERIENCE EQ 4
```

Logical Operator NOT

When the logical operator NOT precedes a condition, the list of qualified records for the condition is replaced by a new list containing similar data records not on the original list, along with any higher level data records that could have descendants similar to the qualified records but do not.

The following three data trees are used in the example for the NOT operator. Arrows point to the qualified records for the following expression:

```
PAY SCHEDULE EQ HOURLY
```

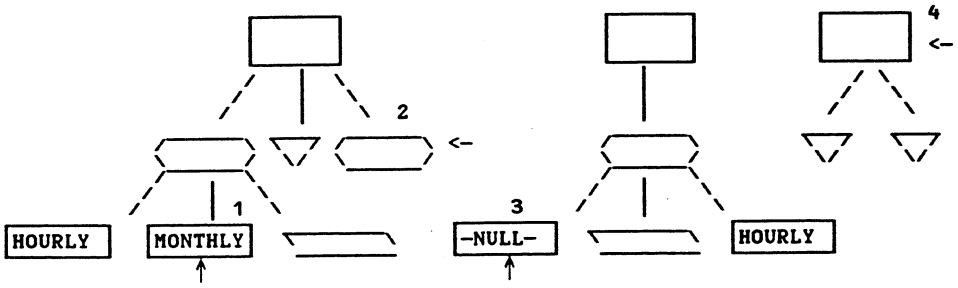


To process the expression

NOT PAY SCHEDULE EQ HOURLY

the software goes through the Hierarchical Table identifying those data records that are similar to the qualified records but are not in the list of qualified records for the condition and also identifying records that could have descendants similar to the qualified records but do not.

The numbers in the following example show identification only. Arrows point to the records qualified by the NOT expression.



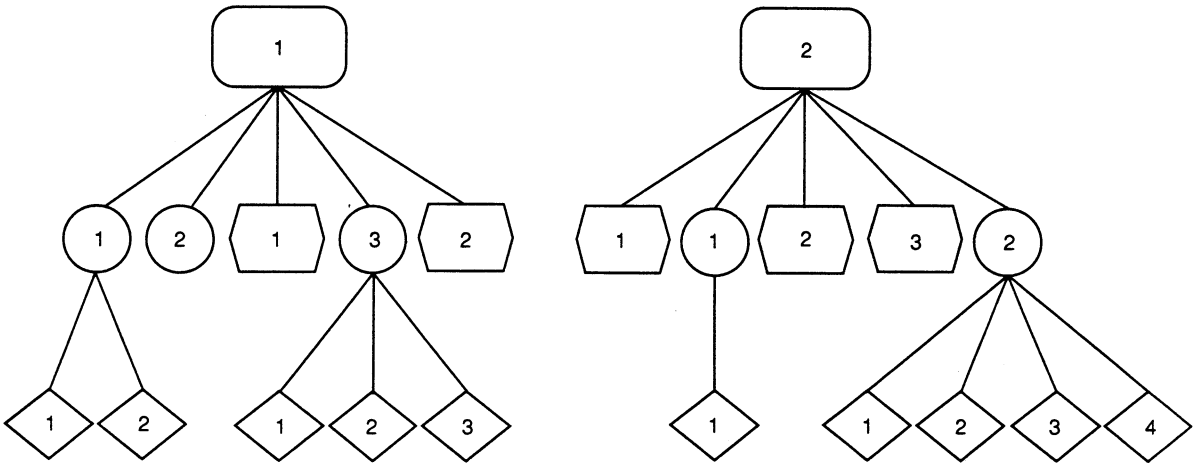
Data record 1 qualifies for the expression because it is similar to the qualified records for the condition, but it is not one of the records that qualify for the condition. The same is true of record 3. Data record 2 could have a descendant similar to the qualified records for the condition, but does not. The same is true of data record 4.

Note: the NOT operator and the NE operator do not produce the same results. The software considers only valued items during processing for NE. The qualified records always contain valued items that do not fulfill the condition requirements. Only record 1 would qualify for the condition PAY SCHEDULE NE HOURLY. Record 3 would not qualify because it does not have a value for the item. Records 2 and 4 would not qualify because only similar records can qualify for a condition with a binary operator.

AT Operator

Using the AT operator in a condition means that for a data record to become qualified, it must not only satisfy the condition but it must also occupy a specific *position* under its parent.

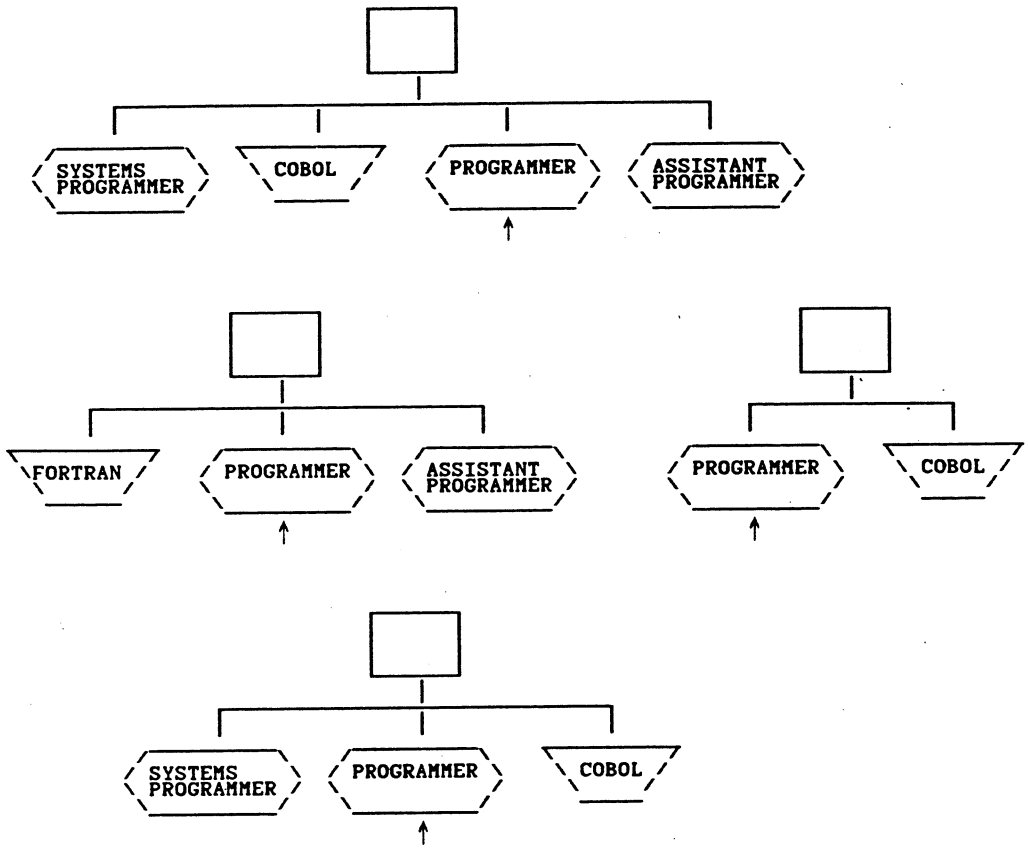
The position of a data record is simply its numbered position in a left-to-right logical order of similar records having the same parent. The following data trees show the position of each data record in each data tree. The number inside each record denotes its position.



The software processes an expression differently according to whether it contains an AT operator. The example shown on these two pages consists of four data trees.

The first part of the example, shown below, consists of a condition without an AT operator. In this case, all records containing PROGRAMMER qualify. Arrows point to the qualified records for the condition

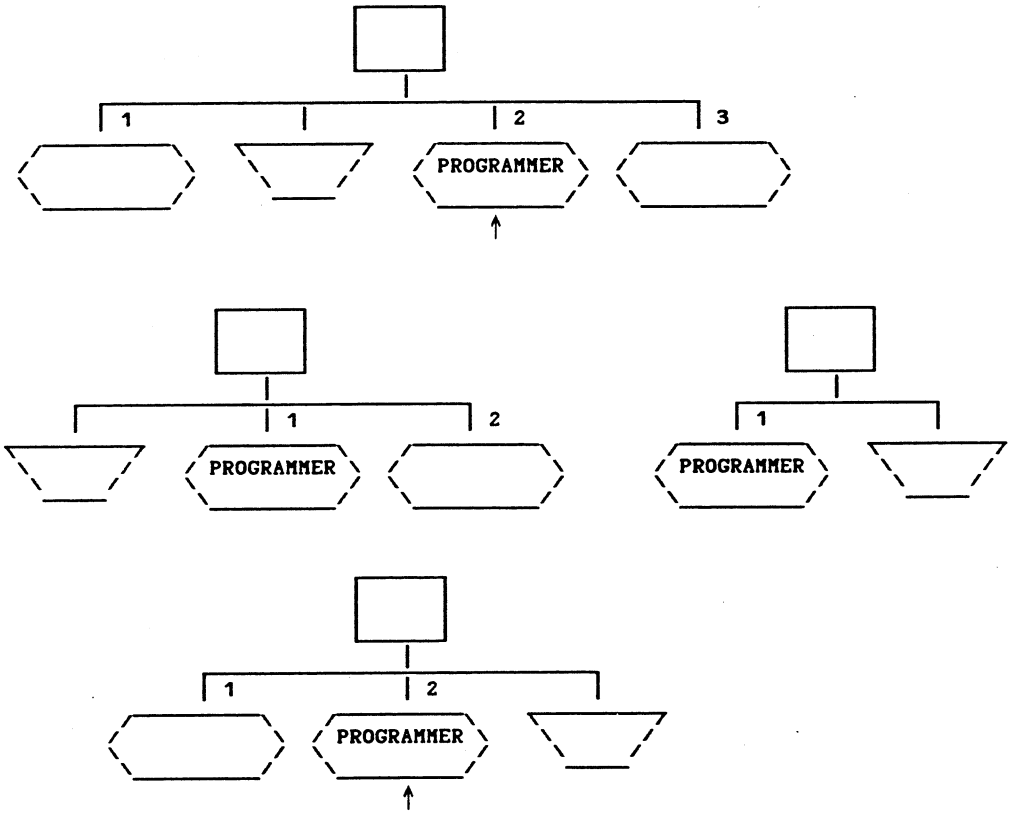
```
POSITION TITLE EQ PROGRAMMER
```



The following expression includes the AT operator, that is, it requests each record that has the value of PROGRAMMER for POSITION TITLE and that also is in the second position under its respective parent record:

```
POSITION TITLE EQ PROGRAMMER AT 2
```

To process AT 2, the software first arranges the qualified records by Hierarchical Table address. Then it goes to the Hierarchical Table entry for the first record. Using internal pointers, it determines whether any of the qualified records are in the second position for that record type. If so, the record remains a qualified records. Arrows indicate the two resulting qualified records.



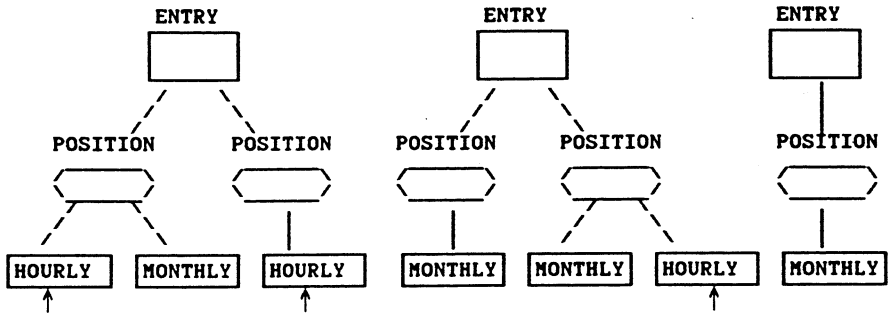
HAS Operator

Preceding a condition with HAS replaces the list of qualified records with a list of ancestor or descendant data records that are occurrences of the specified schema record.

The software processes an expression differently according to whether it contains a HAS operator. The example consists of three data trees.

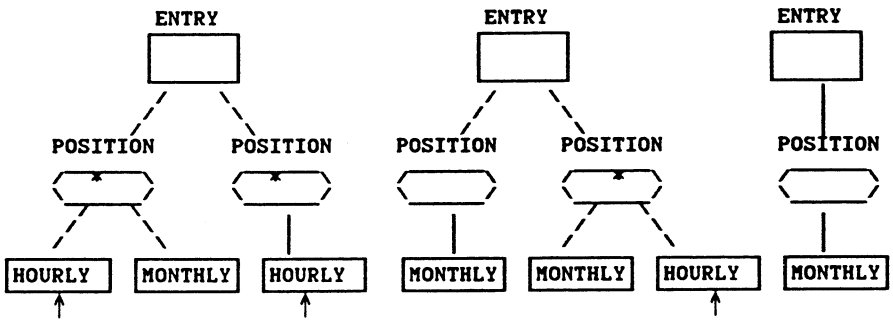
The first part of the example consists of a command that does not contain a HAS operator. Therefore, all records containing the value HOURLY become qualified records. Arrows point to the data records that qualify for the condition

```
PAY SCHEDULE EQ HOURLY
```



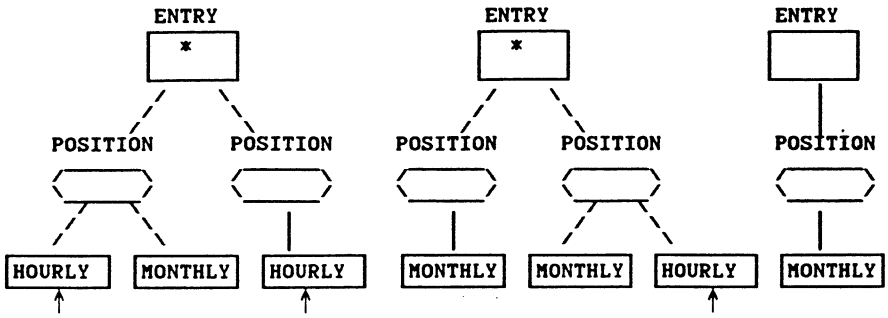
The second part of the example shows an expression that includes a HAS operator. The effect of the HAS operator is to identify some record other than the qualified record as the *focal* record. The focal record determines the family of records that can be retrieved and updated. In this example, the HAS operator identifies the record above the qualified record as the focal record. That is, POSITION WITHIN COMPANY becomes the focal record. (The software uses the Hierarchical Table addresses of the qualified records and follows internal pointers to find the addresses required.) Arrows in the example shown below point to the records that qualify for the condition. The parent records qualified by HAS are marked with asterisks.

```
POSITION WITHIN COMPANY HAS PAY SCHEDULE EQ HOURLY
```



In the example below, the HAS operator raises the focal record to two levels above the qualified record (as opposed to one level in the previous example). The ENTRY records are ancestors of records qualified by the condition. Asterisks mark the ENTRY records that become focal records.

```
ENTRY HAS PAY SCHEDULE EQ HOURLY
```



That is, the HAS operator qualifies only those ancestor records that have a descendant record among the records qualified by the condition.

In summary, the HAS operator replaces the focal record, instead of being the record qualified by the condition. The focal record becomes another specified record, at either a higher or lower level. This process is called *normalization*.

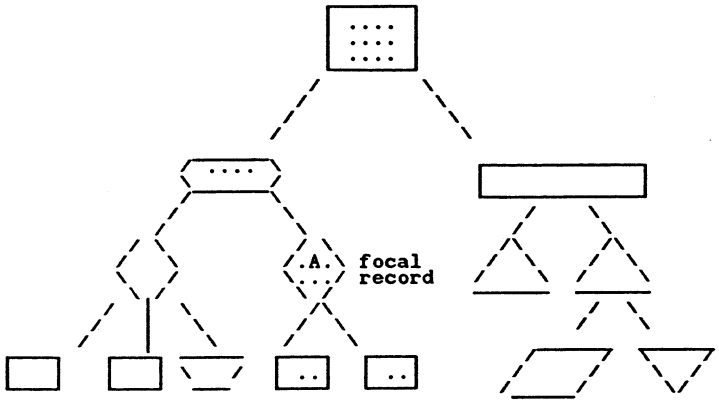
The software uses normalization in two situations:

- to process where-clauses that contain conditions specifying items from different types of schema records
- to retrieve or update data from records that are ancestors or descendants of the qualified data records.

See Chapter 25, "Normalizing with the HAS Operator," for more details.

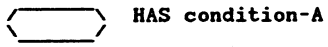
The Importance of Focal records In order for you to retrieve or update data specified in the action-clause, the records containing those data must be in the family of at least one record qualified by the where-clause. To satisfy this requirement you may need to change the focal record by using a HAS operator.

In the following data tree, condition-A qualifies record A. Since the condition contains no HAS operator, record A is the focal record. Therefore, any data record in the family of record A can be accessed. In the illustration below, those records (ancestors and descendants), are marked with dots.

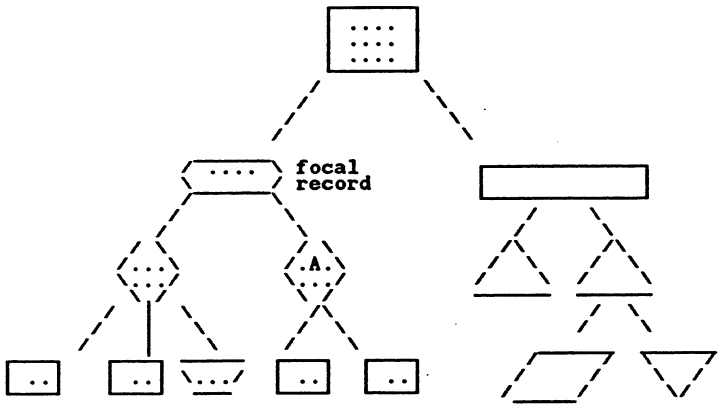


In the data tree below, the HAS operator increases the number of records that can be accessed. In effect, the focal record is raised to a higher level, thereby increasing the number of records that can be accessed by the action-clause.

If the expression is



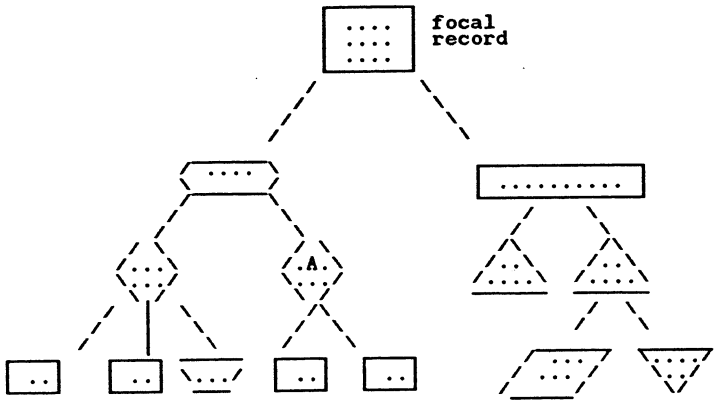
the new focal record is a level higher and the family includes more descendants. Again, the family of records available for access is identified with dots.



If the expression is



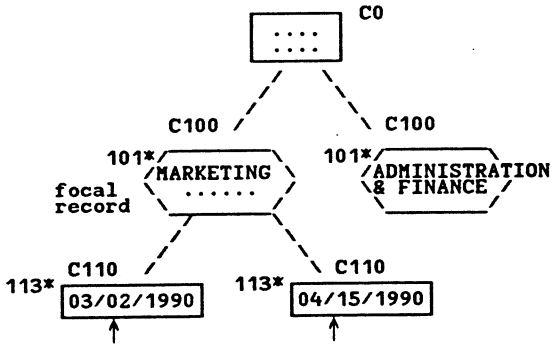
the new focal record is the ENTRY record and the family consists of the whole data tree.



Be sure you decide upon the focal record carefully. Otherwise, a command could yield unwanted results. For example, consider the effect of the two commands discussed next.

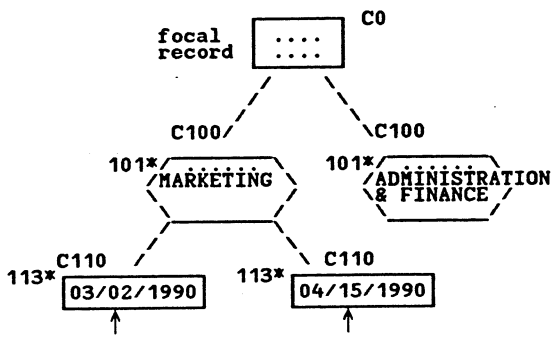
In the command shown below, the focal record is C100. Therefore, the value displayed for C101 is MARKETING, because it appears in a record in the family of the C100 focal record, namely in the focal record itself. (Arrows point to the selected records.)

```
PR C101 WH C100 HAS C113 EQ 03/02/1990 AND C100 HAS C113 EQ 04/15/1990:
```



In contrast, when the command given below is issued, two values (MARKETING and ADMINISTRATION & FINANCE) are selected. They are the values that appear in the C100 records in the family of the C0 focal record. (Arrows point to the selected records.)

```
PR C101 WH C0 HAS C113 EQ 03/02/1990 AND C0 HAS C113 EQ 04/15/1990:
```



In general, the closer the level of the focal record is to the level of the qualified records, the smaller the possibility that the command will select an unwanted record.

The Order of Processing Conditions

When an expression contains more than one operator, the software processes the operators in a specific order:

1. AT
2. HAS
3. NOT
4. AND
5. OR

However, you can change the processing order, as described next.

Controlling the Processing Order with Parentheses

You can control the order of operator processing by using parentheses. Conditions within the innermost pair of parentheses are processed first, followed by those within the surrounding pair, and so forth, moving outward, until all parentheses have been accounted for.

For example, because the software processes the AND operator prior to the OR operator, if you want those employees with an MBA degree and either a major or minor in Marketing, the following command would yield the desired results:

```
PRINT LAST NAME WHERE DEGREE EQ MBA AND (MAJOR FIELD
   EQ MARKETING OR MINOR FIELD EQ MARKETING):
```

On the other hand, if you use the following where-clause, the software would also display those employees who have a minor in Marketing and degrees other than MBAs:

```
WHERE DEGREE EQ MBA AND MAJOR FIELD
   EQ MARKETING OR MINOR FIELD EQ MARKETING
```

In addition to considering processing order, you may need to also consider processing efficiency when you construct a where-clause. To do so, you need to understand how the software uses the index and other tables to process a where-clause.

Efficiency of where-clause processing depends on the placement of conditions within a where-clause, and also on whether conditions contain key or non-key items.

KEY CONDITIONS AND NON-KEY CONDITIONS

To process a condition, the software must first determine whether the item specified in the condition is key or non-key. That is, the software distinguishes between *key-conditions* (those containing key items) and *non-key-conditions* (those containing non-key items).

If the item is key, the software uses the index to make a list of the data records containing values that satisfy the condition. This process is called *key-processing*, and the resulting list consists of the Hierarchical Table addresses of the data records.

If the item is non-key, the software cannot use the index and must search the Data Table to find the qualified records. This process, which is more time consuming, is called *non-key processing*.

However, there are cases in which it is more efficient to use non-key processing, as described in the following topic. If you want to have a key-condition processed as if it were a non-key-condition, you can place the keyword NON-KEY before the condition.

Placement of Conditions

When more than two key-conditions are joined by the same operator, they are processed from right to left, as shown in the example below.

For the expression

condition-x **AND** *condition-y* **AND** *condition-z*

the software first finds those records that satisfy both Condition-Z and Condition-Y, then it finds those records that are also in the list for Condition-X.

Obviously, an AND operation on shorter lists takes less time than on longer lists; there are fewer numbers to match. Also, the list of records found in both lists is always shorter than or equal to either list. Therefore, if the relative sizes of the lists are predictable, it is better to put the conditions producing longer lists in the left part of the expression and those producing shorter lists on the right.

For example, the expression

```
EMPLOYEE STATUS EQ FULL TIME AND GENDER EQ FEMALE AND
   LAST NAME EQ GIBSON
```

is processed faster than the following expression:

```
LAST NAME EQ GIBSON AND GENDER EQ FEMALE AND EMPLOYEE
   STATUS EQ FULL TIME
```

When three or more key conditions joined by AND or OR are being processed from different levels of the data trees, normalization is necessary. Normalization to the next higher or lower level is generally less time consuming than normalization up or down several levels. Therefore, less normalization is required if the conditions are ordered right-to-left by descending level, that is, if the condition closest to level 0 is the rightmost condition. For example, the expression

```
C121 EQ 05/31/1990 AND C112 EQ MONTHLY AND C7 EQ FEMALE
```

is processed faster than the expression

```
C7 EQ FEMALE AND C112 EQ MONTHLY AND C121 EQ 05/31/1990
```

If an expression contains connected key-conditions and you think that one condition will produce a longer list of qualified data records compared with the others, specify that condition as NON-KEY.

NON-KEY requests the software to process a key-condition as a non-key-condition. Although processing a non-key-condition alone is expensive, an expression of the form A AND B where one of A or B is a non-key-condition can be less expensive than the same expression where both A and B are key-conditions.

Suppose A qualifies 400,000 data records and B qualifies only 25 data records. If both A and B are key-conditions, then a lot of time would be spent in creating the long list of qualified data records for A. However, if you specify condition A as non-key, the list for B would be created first; only those 25 data records would be examined in the Data Table for condition A.

Unlike key-conditions, non-key-conditions are processed from left to right. Therefore, for higher efficiency when conditions are joined by AND, the conditions that are likely to qualify small numbers of data records should be placed on the left of those that are likely to qualify large numbers of data records. Since there is no normalization, the order of the levels is arbitrary.

Non-key processing requires a full pass of the Data Table. Since this type of processing can be very expensive in large databases, the DBA or master password holder can prohibit it at your site by issuing the NO FULL PASSES command. If full passes are not allowed at your site, a where-clause that needs a full pass is not processed, and the software ignores the command. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information on the NO FULL PASSES command.)

Chapter 3

Retrieving Data

INTRODUCTION 39

RETRIEVAL OBJECTS 40

RETRIEVAL COMMANDS 41
- *PRINT Command 41*
- *LIST Command 43*
- *COLLECT Command 45*
- *UNLOAD Command 46*
- *TALLY Command 47*

FORMAT OPTIONS 48

ORDERING OUTPUT 49
- *Ordering-Clause 49*
- *By-Clause 50*

INTRODUCTION

QUEST retrieval commands range from very simple to rather complex, giving you a variety of options for organizing and formatting output. The retrieval commands are

- COLLECT
- LIST
- PRINT
- TALLY
- UNLOAD.

These commands must include a *retrieval-clause* that specifies retrieval objects. They can also include an *ordering-clause* that specifies the order in which the data are to be displayed and a *where-clause* that selects only specific data records. In a retrieval command, a retrieval-clause combined with an optional ordering-clause is called an *action-clause*.

```
PRINT LAST NAME, FORENAME, BIRTHDAY, ORDERED BY BIRTHDAY
\_ _ _ _ _retrieval-clause_ _ _ _/ \_ ordering-clause _/
\_ _ _ _ _ _ _ _ _ _ _ _action-clause _ _ _ _ _ _ _ _ _ _/

   WHERE BIRTHDAY GT 01/01/1962:
   \_ _ _ _where-clause_ _ _ _/
```

You can also specify the display format for the data with *format options*.

RETRIEVAL OBJECTS

A retrieval-clause specifies a list of *retrieval objects* to be retrieved by a LIST, PRINT, UNLOAD, or COLLECT command. Retrieval objects can be schema records, schema items, stored functions, stored strings, ad hoc functions, system functions, by-clauses, and Collect File items -- separated by commas.

Therefore, to retrieve data, you must know your database definition, for example, the names or numbers of the components. (Using the DESCRIBE command, you can display a database definition.)

- When specifying the name or number of a schema record, you can indicate either that the values for all items in the record are to be displayed or that the values for all items in the record and its descendant records are to be displayed.

- To retrieve values for a schema item, specify the name or number of the schema item.

- To retrieve Collect File item values, specify the Collect File item name.

- The software allows you to retrieve information that is derived from the stored data. For example, the EMPLOYEE database does not store the average net pay of all employees. But by specifying AVG NET PAY (or AVG C126) in a retrieval command, you can obtain that information. AVG is a *system function*, one of six provided by the software. The others are COUNT, MIN, MAX, SUM, and SIGMA.

- You can specify *ad hoc functions* when you need to do calculations involving data. Ad hoc functions are arithmetic expressions, such as addition, subtraction, multiplication, and division, that combine items, numeric constants, or Collect File items. For example, (REGULAR HOURS + OVERTIME HOURS) yields the total number of hours that an employee has worked in a month.

- You can store frequently used arithmetic expressions in the database definition. This is called a *stored function.*

- If you use a certain list of retrieval objects frequently, you can store the syntax or part of the syntax in the database definition as a *string*. You can then issue the name or component number of the stored string to invoke the stored syntax.

You can also specify several kinds of objects in one retrieval command. For example, you can specify: LAST NAME, FORENAME, AVG NET PAY, and (REGULAR HOURS + OVERTIME HOURS) in one command. And, you can combine ad hoc functions and stored functions with system functions. For example, AVG(REGULAR HOUR + OVERTIME HOURS) yields the average of the total hours worked.

RETRIEVAL COMMANDS

The PRINT, LIST, and UNLOAD commands retrieve any of the retrieval objects mentioned in the previous section. The COLLECT command collects the values for specified items and writes them to a Collect File. The TALLY command retrieves information from data stored in the index.

PRINT Command

The PRINT command displays values in a simple, sequential list. This format makes the PRINT command suitable when the output format is incidental to the query or when large amounts of data are retrieved, for example, entire logical entries.

The following PRINT command displays the last name (C2) and the first name (C3) of the employee whose employee number is 1327:

```
PRINT C2, C3 WHERE C1 EQ 1327:

2* BROOKS
3* RUBEN R.
```

This PRINT command displays the entire logical entry for the same employee.

```
PRINT ENTRY WHERE C1 EQ 1327:
```

(output on next page)

```
1*  1327
2* BROOKS
3* RUBEN R.
4* 05/01/1991
5* 02/25/1964
6* 789-56-2109
7* MALE
8* BLACK
9* FULL TIME
10* 581 XT347
11*   80.00
12*   80.00
14* 2509 LOGANBERRY DR.
15* AUSTIN TX
16* 78744

  101* JR SALES REPRESENTATIVE
  102* MARKETING
  103* MAS
  104* PROFESSIONAL
  105* 05/01/1991

    111*   $2,250.00
    112* MONTHLY
    113* 05/01/1991
    114*     $384.15

      121* 05/31/1991
      122*  184.00
      124*   $2,250.00
      125*     $292.20
      126*   $1,865.85

  201* PASCAL
  202* EXCELLENT
  203*   4

  201* PUBLIC RELATIONS
  202* GOOD
  203*   1

  301* BIRD WATCHING
  303* CONDUCTED SEMINAR AT YWCA 03/15/87

    411* UNIVERSITY OF ILLINOIS AT CHICAGO
    412* BS
    413* 01/07/1985
    414* ENGLISH
    415* MARKETING
```

LIST Command

The LIST command displays data in a columnar format. With LIST, you can also specify a report title, column headings, and a footing line. The output format makes this command suitable for producing reports. You can also store a LIST command as a string to produce periodic reports easily.

The following LIST command is relatively simple and shows how LIST produces clear and easy-to-read output. The output is ordered by the values of item C2 (LAST NAME).

```
LIST C2, C3, C201, ORDERED BY C2 WHERE C201 EQ FORTRAN:

* LAST NAME     FORENAME              SKILL TYPE
***
* FERNANDEZ     SOPHIA                FORTRAN
* GARCIA        FRANCISCO             FORTRAN
* HERNANDEZ     JESSE L.              FORTRAN
* MILLSAP       JOEL B.               FORTRAN
* REDFOX        RICHARD B.            FORTRAN
* REID          DAVID G.              FORTRAN
* SCHOLL        MADISON A.            FORTRAN
```

However, the next LIST command is more complex, in that it uses most of the available syntax.

```
LIST /REPEAT SUPPRESS, TITLE D(5) SALARY REPORT - ADMINISTRATION & FINANCE,
   B(8)--------+EMPLOYEE,B(1)-,B(4)----+NAME,
   B(20)--------------------,S(1),B(5),L(12),B(1),L(10),S(1,2,1),
   B(10),B(10)  DATE  ,B(4),B(7)NET PAY ,S(1),L(1),S(1,1,3),
   B(9),L(10),L(10), S(1),F(30)>>>> SEE NEXT PAGE >>>>/
   C2, C3, C121, C126, ORDERED BY C2, HIGH C121
   WH C102 EQ /ADMINISTRATION & FINANCE/:
```

(output on next page)

```
   SALARY REPORT - ADMINISTRATION & FINANCE     1
                 06/03/1991

***
  --------------------------------
  EMPLOYEE NAME
       CANADY        FRANK A.

              DATE          NET PAY
  *
           05/31/1991     $2,557.00
           04/30/1991     $2,557.00
           03/30/1991     $2,557.00
           02/28/1991     $2,557.00
           01/31/1991     $2,557.00
  --------------------------------
  EMPLOYEE NAME
       COLLINS       LILLIAN

              DATE          NET PAY
  *
           05/31/1991       $772.80
           04/30/1991       $688.80
           03/30/1991       $730.80
           02/28/1991       $646.80
           01/31/1991       $772.80


>>>> SEE NEXT PAGE >>>>


   SALARY REPORT - ADMINISTRATION & FINANCE     2
                 06/03/1991

***
  --------------------------------
  EMPLOYEE NAME
       JONES         RITA M.

              DATE          NET PAY
  *
           05/31/1991      $1,323.80
           04/30/1991      $1,323.80
           03/30/1991      $1,323.80
           02/28/1991      $1,323.80
           01/31/1991      $1,323.80
  --------------------------------
  EMPLOYEE NAME
       KAATZ         FREDDIE

              DATE          NET PAY
  *
           05/31/1991       $772.80
           04/30/1991       $688.80
           03/30/1991       $730.80
           02/28/1991       $646.80
           01/31/1991       $772.80


>>>> SEE NEXT PAGE >>>>
        .
        .
        .
```

COLLECT Command

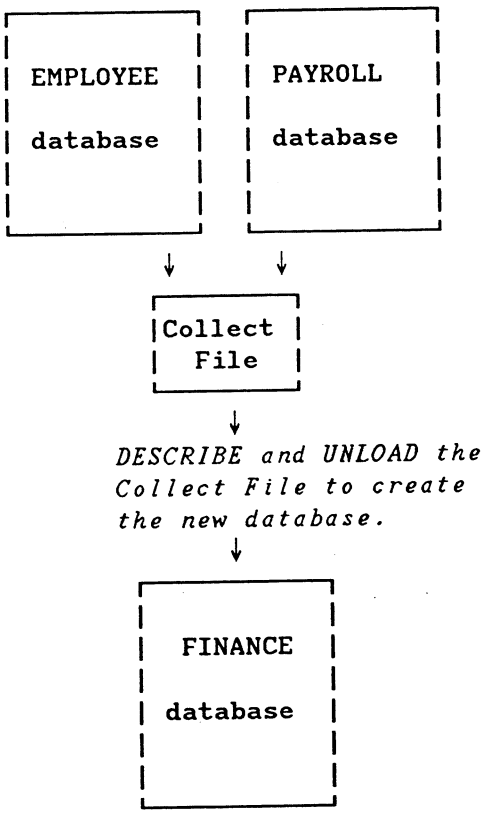
The COLLECT command collects values and writes them to a *Collect File*, which is a relational table. With the COLLECT command, you can gather values from one or more databases and build a sequential file of records that can be used in subsequent QUEST or REPORT language commands within an SCF session.

You can unload a Collect File with the UNLOAD/COLLECT FILE/ command. You can also use the DESCRIBE/CF, DEFINE/ command to produce DEFINE language commands acceptable for defining a new database.

Thus, with the COLLECT, DESCRIBE, and UNLOAD commands, you can gather values from one or more databases and create a new database. The data records in the new database are at level 0; that is, Collect File items become level 0 items in the new database.

The PRINT and LIST commands can display a Collect File. Also, a where-clause can include Collect File items, and the REPORT language can use Collect File items in reports.

The following example shows a method for gathering three items from two databases, creating a Collect File, and then creating a new database from the information contained in the Collect File.




```
USER, DEMO:  DBN IS EMPLOYEE:

(has item EMPLOYEE NUMBER values)

COLLECT EMPLOYEE ID = EMPLOYEE NUMBER
   WHERE DEPT EQ MARKETING:
CONTROL:
USER, ABC:  DBN IS PAYROLL:

(has item EMPNUM values)

COLLECT EMPLOYEE ID, BASE PAY, NET PAY
   WHERE EMPNUM*EQ EMPLOYEE ID*:

REPORT FILE IS X:
DESCRIBE/CF, DEFINE/:
ECHO/MAP:/:
ECHO/COMMAND FILE IS INPUT:/:
REPORT FILE IS Y:
UNLOAD/CF/:
CONTROL:
USER, DEF:
NDB IS FINANCE:
COMMAND FILE IS X:
REPORT FILE IS OUTPUT:
DATA FILE IS Y:
LOAD:
DESCRIBE:
PRINT ENTRY:
```

Notice the ECHO commands that have a string between the slashes. These ECHO commands write supplementary SYSTEM 2000 commands directly to the Report File.

UNLOAD Command

The UNLOAD command retrieves data in a format suitable as input for other SYSTEM 2000 commands. For example, if you need to run tests on a database, it is easier to do so on a subset.

Using the EMPLOYEE database, you could issue the following command to unload entries in the Marketing department to a file:

```
UNLOAD ENTRY WHERE DEPARTMENT EQ MARKETING:
```

You could then load the file into a database with the same schema as the original one. The smaller database is then ready for use as a testing tool.

Although the keyword UNLOAD might give the impression that the command removes data from a database, the command only retrieves data; it does not remove them. The example below shows the output produced by the UNLOAD command.

```
UNLOAD ENTRY WHERE C1 EQ 1327:

1* 1327*2*BROOKS*3*RUBEN R.*4*05/01/1991*5*02/25/1964*6*789-56-2109*7*MALE*8*BLA
CK*9*FULL TIME*10*581 XT347*11* 80.00*12* 80.00*14*2509 LOGANBERRY DR.*15*AUSTIN
 TX*16*78744*100*101*JR SALES REPRESENTATIVE*102*MARKETING*103*MAS*104*PROFESSIO
NAL*105*05/01/1991*110*111*$2250.00*112*MONTHLY*113*05/01/1991*114*$384.15*120*1
21*05/31/1991*122* 184.00*124*$2250.00*125*$292.20*126*$1865.85*200*201*PASCAL*2
02*EXCELLENT*203* 4*200*201*PUBLIC RELATIONS*202*GOOD*203* 1*300*301*BIRD WATCHI
NG*303*CONDUCTED SEMINAR AT YWCA 03/15/87*400*410*411*UNIVERSITY OF ILLINOIS AT
CHICAGO*412*BS*413*01/07/1985*414*ENGLISH*415*MARKETING*END*
 *
```

You can also use the UNLOAD command to unload a Collect File into loader stream format suitable for loading into another database.

TALLY Command

The TALLY command displays information from data stored in the database index. Because the index stores the values for key items, the TALLY command applies only to key items.

For a given key item, TALLY displays each distinct value, the number of times each distinct value occurs, the total number of distinct values, and a count of all the values stored for the item. For example, the following command produces the output below it.

```
TALLY /EACH/ C8:

****************************************
 ITEM-        ETHNIC ORIGIN
****************************************
OCCURRENCES   VALUE
----------------------------------------
        2     AMERICAN INDIAN
        2     ASIAN
        7     BLACK
       31     CAUCASIAN
        8     HISPANIC
----------------------------------------
        5 DISTINCT VALUES
----------------------------------------
       50 TOTAL OCCURRENCES
----------------------------------------
```

FORMAT OPTIONS

Notice that the results produced by the previous sample PRINT commands display each value accompanied by its component number. You can specify that component names be displayed or that the labels be omitted altogether. Notice also how values pertaining to a record are shown together and indented for easier reading. You can also have the data appear in blocks, without indentions. Or you can specify that only values for the specified record be displayed and not the values for descendant records.

These and other options affecting the format of the output are called *format options.*

Some format options, such as the indention option, affect the output of PRINT commands only. Other format options, such as the single or double space option, affect all output. One important option in determining the contents of a database tells the software to print the word -NULL- for null items.

For example, the following command displays component names because the NAME format option is specified:

```
PRINT /NAME/ C410 WHERE C1 EQ 1213:

SCHOOL* UNIVERSITY OF NEBRASKA
```

Contrast the output above with output obtained from the command given next. Here, the use of the NULL format option reveals that educational background data for the employee are missing.

```
PRINT /NULL/ C410 WHERE C1 EQ 1213:

SCHOOL* UNIVERSITY OF NEBRASKA
DEGREE/CERTIFICATE* -NULL-
DATE COMPLETED* -NULL-
MAJOR FIELD* -NULL-
MINOR FIELD* -NULL-
```

The format options available are as follows. (The default settings for each pair are listed in the first column.)

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

You can change format options with the FORMOP, COLLECT, LIST, PRINT, and UNLOAD commands.

ORDERING OUTPUT

Ordering-Clause

When you include a where-clause in a LIST, PRINT, UNLOAD, or COLLECT command, you can specify the order in which the data are to be displayed, unloaded, or collected. That is, you have the option of specifying an *ordering-clause*, which consists of *sort keys*.

Sort keys can be items, records, ad hoc functions, stored functions, or system functions. (You cannot use Collect File items as sort keys.) You can also specify whether the output is to be ordered in ascending or descending order for each sort key.

For example, the output produced by the following command is first sorted by LAST NAME (C2) in ascending order and then by YEARS OF EXPERIENCE (C203) in descending order. The REPEAT SUPPRESS format option suppresses the display of values that would otherwise be repeated.

```
LIST /REPEAT SUPPRESS, TITLE L(12) LAST NAME, FIRST NAMES,
   L(25)       SKILL TYPE, YEARS+EXPERI,B(4) OF+ENCE/
   C2, C3, C201, C203, ORDERED BY C2, HIGH C203
   WHERE C0 HAS C102 EQ EXECUTIVE AT 1:

*  LAST NAME      FIRST NAMES        SKILL TYPE              YEARS OF
                                                            EXPERIENCE
***
* BOWMAN          HUGH E.          RUSSIAN                      15
*                                  PUBLIC RELATIONS             13
*                                  DP SYSTEMS AUDITING          11
*                                  TEACHING                      7
* FAULKNER        CARRIE ANN       TYPING                        6
*                                  SHORTHAND                     5
*                                  GRAPHICS                      1
* GARRETT         OLAN M.          PUBLIC SPEAKING              20
*                                  WRITING                      13
*                                  PUBLIC RELATIONS              8
*                                  FINANCIAL AUDITING            7
* KNAPP           PATRICE R.       CPA                          14
*                                  WRITING                       3
* KNIGHT          ALTHEA           TYPING                        8
*                                  SHORTHAND                     8
*                                  GERMAN                        3
*                                  ACCOUNTING                    2
      .
      .
      .
```

In contrast to the previous example, the following command does not include an ordering-clause, and the REPEAT format option is specified.

```
LIST /REPEAT, TITLE L(12) LAST NAME, FIRST NAMES, L(25)       SKILL TYPE,
   L(6) YEARS+EXPERI,B(4) OF+ENCE/
   C2, C3, C201, C203
   WHERE C0 HAS C102 EQ EXECUTIVE AT 1:
```

(output on next page)

```
*  LAST NAME      FIRST NAMES        SKILL TYPE                YEARS OF
                                                               EXPERIENCE
***
* WATERHOUSE     CLIFTON P.      PUBLIC RELATIONS                19
* WATERHOUSE     CLIFTON P.      FRENCH                          21
* WATERHOUSE     CLIFTON P.      COBOL                           10
* WATERHOUSE     CLIFTON P.      SYSTEMS ANALYSIS & DESIGN       21
* WATERHOUSE     CLIFTON P.      PUBLIC SPEAKING                 12
* WATERHOUSE     CLIFTON P.      ASSEMBLER                        4
* WATERHOUSE     CLIFTON P.      ACCOUNTING                      12
* SALAZAR        YOLANDA         TYPING                          13
* SALAZAR        YOLANDA         ACCOUNTING                       4
* SALAZAR        YOLANDA         SHORTHAND                       10
* BOWMAN         HUGH E.         PUBLIC RELATIONS                13
* BOWMAN         HUGH E.         TEACHING                         7
* BOWMAN         HUGH E.         DP SYSTEMS AUDITING             11
* BOWMAN         HUGH E.         RUSSIAN                         15
* NATHANIEL      DARRYL          PUBLIC RELATIONS                 3
* NATHANIEL      DARRYL          SHORTHAND                        7
* NATHANIEL      DARRYL          TYPING                           9
* NATHANIEL      DARRYL          ACCOUNTING                       9
* GARRETT        OLAN M.         FINANCIAL AUDITING               7
* GARRETT        OLAN M.         PUBLIC SPEAKING                 20
* GARRETT        OLAN M.         PUBLIC RELATIONS                 8
* GARRETT        OLAN M.         WRITING                         13
* KNIGHT         ALTHEA          ETS OPERATOR                     1
      .
      .
      .
```

By-Clause

A retrieval-clause can also include a *by-clause*, which allows you to specify that values within a data tree be displayed together in logical order.

A by-clause consists of a *by-phrase*, which specifies the record at the top of the tree, followed by retrieval objects belonging to the tree.

For example, in the following command, because BY C100 is specified, the values for items in the data trees with C100 records at the top are displayed in logical order and grouped by data tree. Data trees for the two entries are shown in the foldout at the end of this manual.

```
PRINT C2, BY C100, C105, C113, C121 WHERE C1 EQ 1120 OR C1 EQ 1043:

2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 05/31/1991
      121* 04/30/1991
      121* 03/30/1991
      121* 02/28/1991
      121* 01/31/1991
2* REID
  105* 03/01/1989
    113* 03/01/1991
      121* 05/31/1991
      121* 04/30/1991
      121* 03/30/1991
    113* 03/01/1990
      121* 02/28/1991
      121* 01/31/1991
    113* 03/01/1989
2* REID
  105* 02/16/1988
    113* 02/16/1988
```

In contrast, a command that specifies the same items but does not include the BY C100 specification produces the following output. Notice that the values are not grouped by data tree, and they are displayed in physical order. The data record numbers correspond to the order of loading as shown by the data record numbers in the foldout.

```
PR C2, C105, C113, C121 WH C1 EQ 1120 OR C1 EQ 1043:

2* REID
  105* 03/01/1989
    113* 03/01/1989     ← from data record 15
2* REID
  105* 02/16/1988
    113* 02/16/1988     ← from data record 17
2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 01/31/1991   ← from data record 34
2* REID
  105* 03/01/1989
    113* 03/01/1990
      121* 01/31/1991   ← from data record 35
2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 02/28/1991   ← from data record 36
2* REID
  105* 03/01/1989
    113* 03/01/1990
      121* 02/28/1991   ← from data record 37
2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 03/30/1991   ← from data record 39
2* REID
  105* 03/01/1989
    113* 03/01/1991
      121* 03/30/1991   ← from data record 40
2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 04/30/1991   ← from data record 41
2* REID
  105* 03/01/1989
    113* 03/01/1991
      121* 04/30/1991   ← from data record 42
2* GIBSON
  105* 10/01/1990
    113* 10/01/1990
      121* 05/31/1991   ← from data record 44
2* REID
  105* 03/01/1989
    113* 03/01/1991
      121* 05/31/1991   ← from data record 45
```

Chapter 4

Updating Data

INTRODUCTION 53

COMMANDS FOR UPDATING ITEM VALUES 54
- *ADD Command 54*
- *CHANGE Command 56*
- *REMOVE Command 58*
- *ASSIGN Command 59*

COMMANDS FOR UPDATING DATA TREES 61
- *INSERT TREE Command 61*
- *REMOVE TREE Command 64*
- *ASSIGN TREE Command 66*
- *MOVE TREE Command 67*
- *COPY TREE Command 68*

INTRODUCTION

QUEST language update commands allow you to add, change, remove, move, or copy data.

You need to specify which occurrences of an item or record are to be updated. The record or the record containing the specified item is called the *target record* for the command. There are three ways to specify which occurrences of a record or item are to be modified:

- trace notation to specify their position. *Trace notation* points to a specific data record in a tree, by identifying the exact data record at each level within a logical entry.
- a where-clause to specify criteria that values in a data tree must meet for the update to occur.
- both a where-clause and trace notation.

If you include only a where-clause, the records to be updated (that is, the selected data records) are those occurrences of the target record that are related to the qualified data records. Trace notation determines the selected data records by their position in the database.

If you include both trace notation and a where-clause, the selected data records are determined by both their position and their relationship to the qualified data records.

You can also specify the position in which a new record or data tree is to be inserted in the database.

COMMANDS FOR UPDATING ITEM VALUES

The commands for updating item values affect only records that currently exist in the database. The four commands are: ADD, CHANGE, REMOVE, and ASSIGN. You cannot use any of these commands to insert or remove data records.

For example, suppose the following data record exists:

```
111* $1,800.00
112* MONTHLY
113* 03/01/1990
114* -NULL-
```

You could use one update command to add a value for C114, another to change the value for C111, and still another to remove the value for C113. Or you could make all these changes at once.

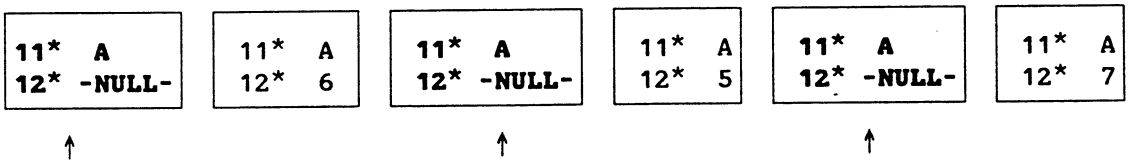
ADD Command

The ADD command adds values only where the specified items are not currently valued. The ADD command enters a value for a single null item or enters values for several different null items in selected data records.

You cannot use Collect File items in the action-clause of an ADD command. However, they can be used in a where-clause.

The example below the following command shows the data records selected:

```
ADD C12 EQ 5* WHERE C11 EQ A:
```



When the command is executed, the software issues the following message, and the specified value (that is, 5) is entered in the records marked with arrows. Notice that the message indicates the number of selected records, not the number of records where the value was entered.

```
-342-        6 SELECTED DATA RECORD(S) -
```

When the ADD command specifies an item, you can also enter the result of a computation. For example, the following command calculates the value for NET PAY and enters the value in each selected record with a null NET PAY.

```
ADD NET PAY = (GROSS PAY - CURRENT DEDUCTION) WH C121 EQ 05/31/1990:
```

Example 1

This command adds one new value in a data record:

```
ADD C412 EQ BS* WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* UNIV. OF MICHIGAN
412* -NULL-	412* BS
413* 06/12/1981	413* 06/12/1981
414* -NULL-	414* -NULL-
415* -NULL-	415* -NULL-

Example 2

This command enters several new values in a data record. Because values for several items are to be added, the record is specified (that is, C410). Notice that the value for C412 does not change even though BA is specified in the command. (BA would have been entered only if C412 were null.)

```
ADD C410 EQ 412*BA*414*MATH*415*ECON*END* WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* UNIV. OF MICHIGAN
412* BS	412* BS
413* 06/12/1981	413* 06/12/1981
414* -NULL-	414* MATH
415* -NULL-	415* ECON

The output was produced with this command:

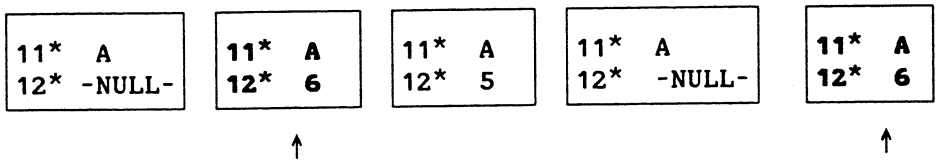
```
PRINT/NULL/ C410 WHERE C1 EQ 1005:
```

CHANGE Command

The CHANGE command replaces values only where the specified items are currently valued. This command replaces a single value or several values in selected data records.

This example shows the five data records selected for the command:

```
CHANGE C12 EQ 5* WHERE C11 EQ A:
```



When the command is executed, the software issues the message shown below, and the specified value (that is, 5) replaces the current value for C12. Arrows point to the records where the change takes place. Notice that the message indicates the number of records selected, not the number of records where the value was replaced.

```
-342-          5 SELECTED DATA RECORD(S) -
```

When the CHANGE command specifies an item, you can also replace a value with the result of a computation. For example, the following command replaces the current value for GROSS PAY with a value that is $100.00 larger.

```
CHANGE GROSS PAY = (100. + GROSS PAY) WHERE C121 EQ 05/31/1990:
```

You cannot use Collect File items in the action-clause of a CHANGE command.

Example 3

This command replaces one value in a data record.

```
CHANGE C412 EQ BA* WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* UNIV. OF MICHIGAN
412* BS	412* BA
413* 06/12/1981	413* 06/12/1981
414* MATH	414* MATH
415* -NULL-	415* -NULL-

Example 4

This command replaces several values in a data record. Since the values for several items are to be replaced, the record is specified (C410). Notice that C415 remains null even though ECONOMICS is specified in the command. The replacement would have taken place only if C415 were already valued.

```
CHANGE C410 EQ 412*BS*414*MATHEMATICS*415*ECONOMICS*END*
   WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* UNIV. OF MICHIGAN
412* BA	412* BS
413* 06/12/1981	413* 06/12/1981
414* MATH	414* MATHEMATICS
415* -NULL-	415* -NULL-

The output was produced with this command:

```
PRINT/NULL/ C410 WHERE C1 EQ 1005:
```

REMOVE Command

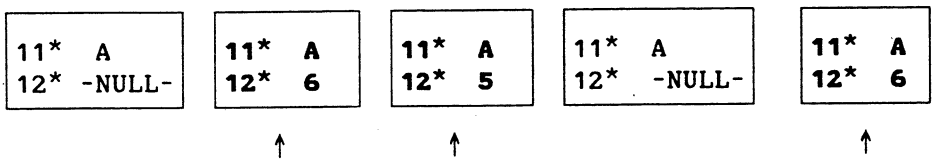
The REMOVE command replaces values with nulls and does not affect items that are already null. The REMOVE command removes the value of a single item or all items from selected data records.

Note: If you need to remove more than one value but not all values, you must issue a REMOVE command for each separate value.

You cannot use Collect File items in the action-clause of a REMOVE command. However, they can be used in a where-clause.

The following example shows the five data records selected by the command:

```
REMOVE C12 WHERE C11 EQ A:
```



When the command is executed, the software issues the message shown below and removes the values from the three C12 items that are valued. Arrows point to the records affected by the command. Notice that the message indicates the number of records selected, not the number of records where a value was removed.

```
-342-        5 SELECTED DATA RECORD(S) -
```

Example 5

This command removes the value of one item in a data record.

```
REMOVE C413 WHERE C1 EQ 1005:
```

Data record before command is executed

```
411* UNIV. OF MICHIGAN
412* BS
413* 06/12/1981
414* MATHEMATICS
415* ECONOMICS
```

Data record after command is executed

```
411* UNIV. OF MICHIGAN
412* BS
413* -NULL-
414* MATHEMATICS
415* ECONOMICS
```

Example 6

This command removes the values for all items in a data record. Since all values are to be removed, the record (C410) is specified in the command.

```
REMOVE C410 WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* -NULL-
412* BS	412* -NULL-
413* -NULL-	413* -NULL-
414* MATHEMATICS	414* -NULL-
415* ECONOMICS	415* -NULL-

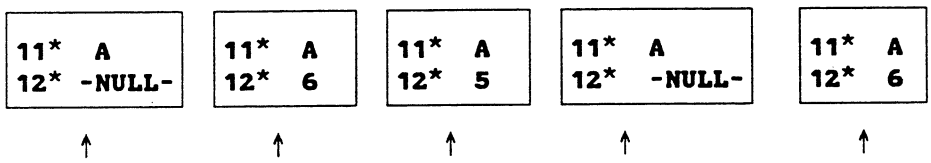
ASSIGN Command

The ASSIGN command assigns new values to items regardless of whether the items involved are already valued. The ASSIGN command assigns either one value to a single item or multiple values to several items in selected data records. (In effect, the ASSIGN command is equivalent to a REMOVE command followed by an ADD command.)

You cannot use Collect File items in the action-clause of an ASSIGN command.

The following example shows the five data records selected by the command:

```
ASSIGN C12 EQ 5* WHERE C11 EQ A:
```



When the command is executed, the software issues the message shown below, and the specified value (that is, 5) replaces the current values for C12 or is added to the records where C12 is null. Arrows point to the records where the new value is assigned. Notice that the number of data records selected coincides with the number of records affected by the command.

```
-342-          5 SELECTED DATA RECORD(S) -
```

When the ASSIGN command specifies an item, you can also assign a value that is the result of a computation. For example, the following command calculates the value for NET PAY and assigns that value in each selected data record:

```
ASSIGN NET PAY = (GROSS PAY - CURRENT DEDUCTION)
   WHERE C121 EQ 05/31/1991:
```

The output was produced with this command:

```
PRINT/NULL/ C410 WHERE C1 EQ 1005:
```

Example 7

This command assigns one new value to a data record.

```
ASSIGN C411 EQ UNIV. OF MICHIGAN* WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF NICHIGAN	411* UNIV. OF MICHIGAN
412* BA	412* BA
413* 06/12/1981	413* 06/12/1981
414* -NULL-	414* -NULL-
415* -NULL-	415* -NULL-

Example 8

This command enters several new values in a data record. Since several new values are to be assigned, the record (C410) is specified in the command. Notice that the values for C411 and C412 remain unchanged, the value for C413 is removed and no value assigned to it, and new values are assigned to C414 and C415.

```
ASSIGN C410 EQ 411*UNIV. OF MICHIGAN*412*BA*414*ECONOMICS*
   415*MATHEMATICS*END* WHERE C1 EQ 1005:
```

Data record before command is executed	Data record after command is executed
411* UNIV. OF MICHIGAN	411* UNIV. OF MICHIGAN
412* BA	412* BA
413* 06/12/1981	413* -NULL-
414* -NULL-	414* ECONOMICS
415* -NULL-	415* MATHEMATICS

The output was produced with this command:

```
PRINT/NULL/ C410 WHERE C1 EQ 1005:
```

COMMANDS FOR UPDATING DATA TREES

The commands to update data trees affect the structure of the database. That is, they either insert new data records, replace data records, remove data records, move data records, or copy data records from one position to another. The commands are: INSERT TREE, ASSIGN TREE, REMOVE TREE, MOVE TREE, and COPY TREE.

INSERT TREE Command

The INSERT TREE command inserts new data records where none exist.

You cannot use Collect File items in the action-clause of an INSERT TREE command.

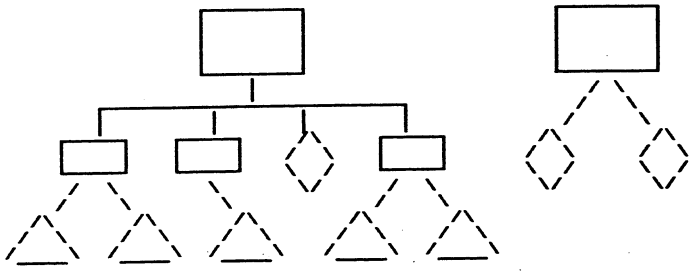
The following INSERT TREE command inserts new training information for an employee:

```
IT C420 EQ 421*IBM*422*COBOL*423*09/09/1990*END* AFTER C2
   EQ REID AND C421 EQ TECNIK:
```

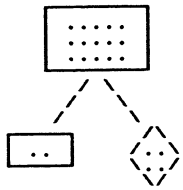
The following examples show the positions where INSERT TREE commands can insert new data trees. The placement of inserted data records and data trees depends on the where-clause, the use of BEFORE and AFTER, and the trace notation used, if any.

Example 1

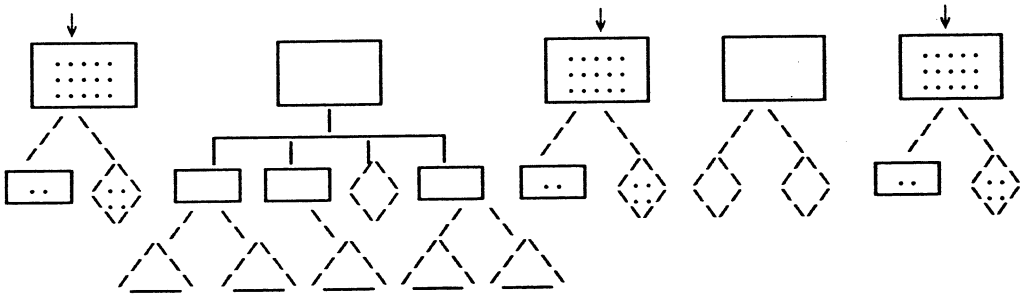
Assume a database with only two logical entries, as shown below:



Suppose the data tree to be inserted is another logical entry, such as the one shown below,

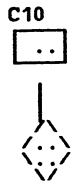


It can be inserted in any of the three positions indicated below by using commands of the form INSERT TREE C0 ... .

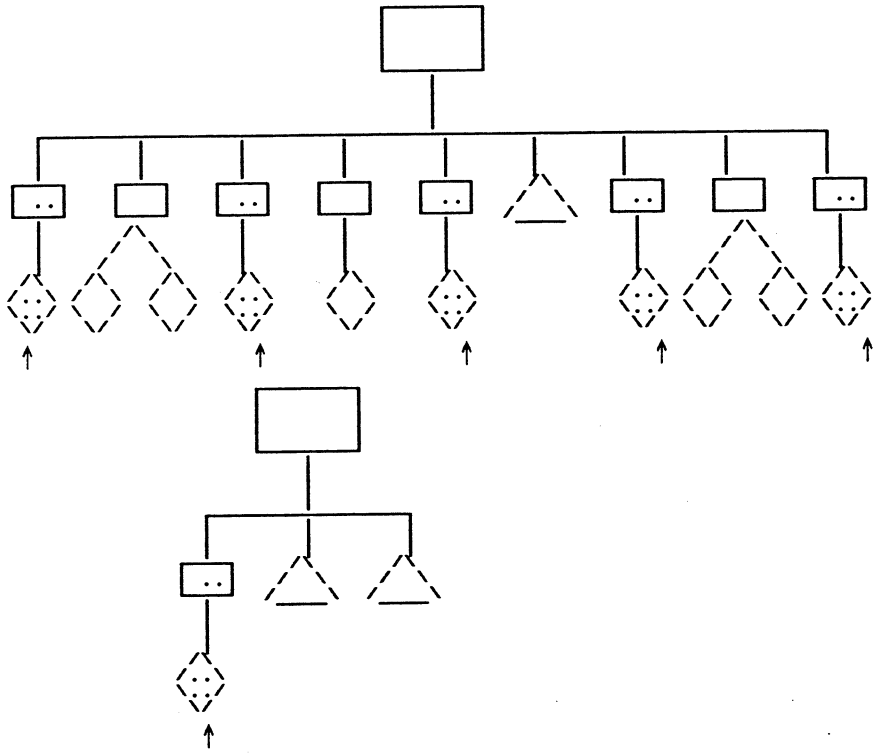


Example 2

In this example, suppose the data tree to be inserted has a non-level 0 record at the top.



It can be inserted in the positions indicated below by using commands of the form INSERT TREE C10 ... .



The syntax of the INSERT TREE command determines where the data tree is inserted. You can also use INSERT TREE commands to insert new data records, that is, data records without descendants.

REMOVE TREE Command

The REMOVE TREE command removes data records and their descendants. You cannot use Collect File items in the action-clause of a REMOVE TREE command.

The following command removes the entire logical entry for the employee with number 1400:

```
REMOVE TREE C0 WHERE C2 EQ 1400:
```

The following command removes all educational background data for that employee:

```
REMOVE TREE C400 WHERE C2 EQ 1400:
```

It is important to understand the difference between the REMOVE command and the REMOVE TREE command.

The following REMOVE command removes the values from selected C100 data records; it does not remove the records from the database.

```
REMOVE C100 WHERE C2 EQ 1400:
```

Example 3 shows the difference between the REMOVE command and the REMOVE TREE command.

Example 3

```
REMOVE C100 WHERE C102 EQ MARKETING AND C1 EQ 1250:

   -342-     1 SELECTED RECORD(S) -
```

Data record before command is executed

```
1*  1250

  101* SECRETARY
  102* EXECUTIVE
  103* OMG

    111*       $800.00
    112* MONTHLY

    111*       $780.00
    112* MONTHLY

  101* SECRETARY
  102* MARKETING
  103* VPB

    111*       $750.00
    112* MONTHLY

    111*       $730.00
    112* MONTHLY
```

Data record after command is executed

```
1*  1250

  101* SECRETARY
  102* EXECUTIVE
  103* OMG

    111*       $800.00
    112* MONTHLY

    111*       $780.00
    112* MONTHLY

  101* -NULL-
  102* -NULL-
  103* -NULL-

    111*       $750.00
    112* MONTHLY

    111*       $730.00
    112* MONTHLY
```

Example 3 (continued)

```
REMOVE TREE C100 WHERE C102 EQ MARKETING AND C1 EQ 1250:

   -342-     1 SELECTED RECORD(S) -
```

Data record before command is executed

```
1*  1250

  101* SECRETARY
  102* EXECUTIVE
  103* OMG

    111*      $800.00
    112* MONTHLY

    111*      $780.00
    112* MONTHLY

  101* SECRETARY
  102* MARKETING
  103* VPB

    111*      $750.00
    112* MONTHLY

    111*      $730.00
    112* MONTHLY
```

Data record after command is executed

```
1*  1250

  101* SECRETARY
  102* EXECUTIVE
  103* OMG

    111*      $800.00
    112* MONTHLY

    111*      $780.00
    112* MONTHLY
```

You can see the difference between a REMOVE command and a REMOVE TREE command in the effect that each has on the Hierarchical Table and on the Data Table. With a REMOVE TREE command, the space in the Data Table (made available by the removal of a record) becomes part of reusable space and can be reused by any new record of the same type. Also, once a record is removed by a REMOVE TREE command, the software modifies the entries in the Hierarchical Table, closing the gap created by the removal.

For example, if data records A, B, and C are siblings and record B is removed, the Hierarchical Table entry for A would indicate record C as its right sibling.

The output was produced with this command:

```
PRINT/NULL/C1, C100 WHERE C1 EQ 1250:
```

ASSIGN TREE Command

The ASSIGN TREE command replaces data trees. The command removes selected data records and their descendants and inserts new data records and any descendants in the same logical position as the selected data records.

You cannot use Collect File items in the action-clause of an ASSIGN TREE command.

Example 4

```
ASSIGN TREE C10 EQ 11*2*12*BB*100*101*BB1*END*WHERE C1 EQ 1000
   AND C11 EQ 2:
```

Data record before command is executed

```
1*  1000
2*  A

  11* 1
  12* AA

  11* 2
  12* AB

    101*  AA1

    101*  AA2
```

Data record after command is executed

```
1*  1000
2*  A

  11* 1
  12* AA

  11* 2
  12* BB

    101*  BB1
```

MOVE TREE Command

The MOVE TREE command moves a data tree from one position in the database to another. The move is a logical move in that only the data record pointers are changed in the Hierarchical Table; the physical records in the Data Table and the indexes do not require any changes.

You can move an entire logical entry from one place to another or move a subtree from one entry to another. This is not a copy operation. The selected top node and all descendants are moved to a new position. The MOVE TREE command is comparable to unloading a data tree, removing the tree, and inserting it at a new place in the database. However, MOVE TREE is much faster because only the logical pointers are changed; the physical data and records are not disturbed.

The MOVE TREE command contains two where-clauses. The first where-clause must select one top node of the tree to be moved; it is the "from" where-clause. The second where-clause must establish one point of insertion for the tree being moved; it is the "to" where-clause. You can also use a partial trace with either where-clause or a full trace to establish the "from" and "to" positions.

Example 5

The following MOVE TREE commands move entire logical entries:

```
MOVE TREE ENTRY WHERE C1 EQ A1 TO AFTER C1 EQ A2:

MOVE TREE C0 *1 TO BEFORE C11 EQ B2:

MOVE TREE ENTRY WHERE C1 EQ A1 TO *3:
```

Example 6

These MOVE TREE commands move subtrees:

```
MOVE TREE C20 WHERE C21 EQ C1 TO BEFORE C21 EQ C2:

MOVE TREE C10 *1*1 TO *0*2:

MOVE TREE C10 *1 WHERE C1 EQ A3 TO *0 WHERE SAME:
```

COPY TREE Command

The COPY TREE command copies a data record and its descendants from one position in the database to another. The data tree to be copied can be an entire logical entry or any subtree in a logical entry. Copying a data tree causes changes in the Hierarchical Table, in the Data Table, and, if key values are copied, in the index tables.

The copy operation requires a one-to-one relationship. That is, a data tree is copied from its old position to a single new point of attachment. It cannot be copied to multiple new positions.

Example 7

The following COPY TREE commands copy entire logical entries:

```
COPY TREE ENTRY WHERE C1 EQ A1 TO AFTER C1 EQ A2:

COPY TREE C0 *1 TO BEFORE C11 EQ B2:

COPY TREE C0 WH C21 EQ D1 TO BEFORE C1 EQ A3:
```

Chapter 5

Supplementary Capabilities for Retrieving and Updating

CONTROLLING THE PROCESSING OF COMMANDS 70
- *LIMIT Command 70*
- *STOP IF and STOP AFTER SCAN Commands 70*
- *ENABLE EXECUTION and DISABLE EXECUTION Commands 70*
- *CLEAR Commands 70*

REPEATING COMMAND UNITS 71
- *DITTO Operator 71*
- *SAME Operator 71*
- *PREVIOUS Operator 72*

ACCESSING DERIVED INFORMATION 72
- *System Functions 72*
- *Ad Hoc Functions 74*
- *System Strings 75*

STORED STRINGS AND FUNCTIONS 75
- *Using Stored Strings 75*
- *Using Stored Functions 79*

RELOADING A DATABASE 80

This chapter presents capabilities of the QUEST language that make using it more convenient for you.

CONTROLLING THE PROCESSING OF COMMANDS

The QUEST language includes several commands that allow you to control the processing of retrieval and update commands.

LIMIT Command

The LIMIT command allows you to set limits on the number of data records that are affected by retrieval or update commands.

STOP IF and STOP AFTER SCAN Commands

The STOP IF and STOP AFTER SCAN commands allow you to specify an upper limit of errors in a job. The CONTINUE IF and CONTINUE AFTER SCAN commands countermand the respective STOP command within a job.

ENABLE EXECUTION and DISABLE EXECUTION Commands

The ENABLE EXECUTION command instructs the software to scan retrieval and update commands so that only syntactically correct commands are executed. The DISABLE EXECUTION command prevents the execution of all update and retrieval commands, but it does allow the software to check for syntax errors.

CLEAR Commands

The CLEAR commands allow you to control when memory-resident pages modified by update operations are to be written to disk.

REPEATING COMMAND UNITS

The following QUEST operators simplify repetitive processing; they repeat portions of previously executed commands.

DITTO Operator

The DITTO operator repeats the action-clause of the most recently executed retrieval or update command. DITTO allows you to issue several retrieval or update commands with the same action-clause but different where-clauses. For example, suppose you issue this command:

```
LIST /REPEAT SUPPRESS/ C1, C2, C3, BY C100, C101, C102, C105,
   ORDERED BY C2, C101, C105 WHERE C8 EQ HISPANIC:
```

Then, when you issue the following command, the software uses the action-clause of the previous command:

```
DITTO WHERE C8 EQ BLACK:
```

SAME Operator

The SAME operator retains and uses the list of qualified data records generated by the where-clause of a previously executed retrieval or update command.

For example, the following commands show one use of the SAME operator:

```
PR C120 WH C1 EQ 1050 AND C101 EXISTS AT 1 AND C111 EXISTS AT 1:

CHANGE C126 EQ $5220.40* WHERE SAME AND C121 EXISTS AT 1:

PRINT C120 WHERE SAME:
```

The first command displays the C120 records that are qualified by its where-clause. The second command takes those qualified records (by specifying WHERE SAME) and imposes another condition on them (by specifying AND C121 EXIST AT 1). The third command, by specifying WHERE SAME, prints the new group of qualified records without any further processing being required.

Without the SAME operator, the where-clause in the second and third commands would have to be

```
WHERE C1 EQ 1050 AND C101 EXISTS AT 1 AND C111 EXISTS AT 1
   AND C121 EXISTS AT 1
```

PREVIOUS Operator

The PREVIOUS operator repeats the value stream contained in the action-clause of the most recently executed update command.

For example, suppose you issue the following command to add educational background data into a record where C1 EQ 1250:

```
ADD C410 EQ 411*UNIVERSITY OF SOUTHERN CALIFORNIA*412*BS*414
   *COMPUTER SCIENCES*END* WHERE C1 EQ 1250:
```

If the same data need to be changed in the C410 record where C1 EQ 1260, you can use the PREVIOUS operator in a CHANGE command, as shown below:

```
CHANGE C410 EQ PREVIOUS WHERE C1 EQ 1260:
```

ACCESSING DERIVED INFORMATION

SYSTEM 2000 software allows you to retrieve information derived from stored data by performing calculations on the data. The results of calculations can also be stored in the database with update commands.

System Functions

System functions provide statistics on the values stored in the database. There are six system functions:

- COUNT counts the number of times a record or item occurs.
- SUM adds the values of numeric items.
- MAX finds the maximum value of an item.
- MIN finds the minimum value of an item.
- AVG finds the average value of a numeric item.
- SIGMA computes the standard deviation of the values for a numeric item.

The following command retrieves the values for REGULAR HOURS (C122) as well as statistics on those values for the employee with number 1261.

```
PRINT /NAME/ C122, COUNT C122, MAX C122, MIN C122, SUM C122,
   AVG C122, SIGMA C122 WHERE C1 EQ 1261:
```

The output appears on the following page. Note that system function values are displayed before item values.

```
CNT REGULAR HOURS*  5  1
MAX REGULAR HOURS*  184.00  2
MIN REGULAR HOURS*  160.00  3
SUM REGULAR HOURS*  872.000  4
AVG REGULAR HOURS*  174.400  5
SIG REGULAR HOURS*  10.431  6
    REGULAR HOURS*  184.00
    REGULAR HOURS*  168.00
    REGULAR HOURS*  176.00
    REGULAR HOURS*  160.00
    REGULAR HOURS*  184.00
```

1 Count of C122 values for C1 EQ 1261: 5.

2 Largest of those values: 184.00.

3 Smallest of those values: 160.00.

4 Sum of those values: 872.00

```
   184.00
   168.00
   176.00
   160.00
+  184.00
  -------
   872.00
```

5 Average of the values: 872.00/5 = 174.400.

6 Standard deviation of the values:

$$\sqrt{(152512.00 - 872.00^2/5)/(5\text{-}1)} = 10.4307 \approx 10.431$$

where

$$152512.00 = (184.00)^2 + (168.00)^2 + (176.00)^2 + (160.00)^2 + (184.00)^2$$

Ad Hoc Functions

Ad hoc functions obtain calculations based on the data. An ad hoc function is an arithmetic expression. You can specify an ad hoc function in a retrieval-clause, an ordering-clause, or the action-clause of an update command.

Ad hoc functions can include system functions, and system functions can include ad hoc functions. Also, you can store ad hoc functions in the database definition so that they can be assigned a component number, a component name, and an optional data type. An ad hoc function stored in the definition is called a stored function.

Example 1

The following command converts stored values for temperatures in degrees Fahrenheit (C134) to temperatures in degrees Centigrade and adds the resulting values to item C135 in selected data records. Notice that the ad hoc function is enclosed in parentheses.

```
ADD C135 = ((C134 - 32.0)*5/9) WHERE C134 EXISTS:
```

Example 2

The following command displays the values for FEDERAL TAX DEDUCTION (C125), NET PAY (C126), and the percentage of C126 that C125 represents.

```
LIST C125, C126, ((C125 / C126) * 100) WHERE C126 EXISTS AT 1:
```

Example 3

This command displays the last name, first name, and age of each employee in the Marketing Department. (*FTODAY* - BIRTHDAY) yields the number of days that have elapsed since the employee's birthday, and dividing by 365.25 gives the age in years.

```
LIST LAST NAME, FORENAME, ((*FTODAY* - BIRTHDAY)/365.25)
   WHERE DEPARTMENT EQ MARKETING:
```

System Strings

System strings provide you the current date and time according to the computer's internal calendar and clock. The system string *TODAY* provides the current date in date format, *FTODAY* provides the current date in function format, and *NOW* provides the current time in hours, minutes, and seconds.

Example 4

The following command displays the PART NUMBER and SUPPLIER of each part delivered today.

```
LIST PART NUMBER, SUPPLIER WHERE DELIVERY DATE EQ *TODAY*:
```

Example 5

The following command adds the time of delivery of part number 14629.

```
ADD DELIVERY TIME EQ *NOW** WHERE PART NUMBER EQ 14629:
```

STORED STRINGS AND FUNCTIONS

A *stored string* is a command, a part of a command, or a series of commands stored as a component in a database definition and invoked through the QUEST language when you issue the string name or number. A *stored function* is an ad hoc function stored as a component in a database definition and invoked through the QUEST language when you issue the function name or number.

Stored strings and functions can be simple or parametric. Simple strings and functions are invoked by their component name or number. Parametric strings and functions require that you supply values along with the component name or number.

Because stored strings and functions are part of the database definition, they must be defined by the master password holder.

Using Stored Strings

Stored strings are particularly useful when you need a lengthy operation frequently and want to refer to it by a short code word. Using a string to stand for a command or a series of commands relieves you of formulating, coding, and possibly debugging the same command or set of commands every time the operation is needed. Also, invoking a string amounts to issuing a component name or number and possibly supplying some values. You can define strings to custom fit the reporting and updating requirements of different users of the database. Thus, users can access the database even if they have little knowledge of the software.

The following examples show how you can use stored strings to simplify using a retrieval command.

Example 1

Consider the following LIST command:

```
LIST/TITLE D(23)MONTHLY REPORT,B(10),B(12)DEPARTMENT--,B(30)
   MARKETING, S(2),L(6)EMPLOY+ NUMBE,B(5)EE+R,L(18)LAST NAME,L(16)FIRST
   NAME L(10)NET PAY,S(1)/C1,C2,C3,C126 WHERE C121 EQ 03/30/91
   AND C102 EQ MARKETING:
```

You could store the command in a string during a DEFINE session, as follows:

```
DEFINE:
1115* REPORT (STRING $LIST/TITLE D(23)MONTHLY REPORT,B(10),
   B(12)DEPARTMENT--,B(30)
   MARKETING, S(2),L(6)EMPLOY+ NUMBE,B(5)EE+R,L(18)LAST NAME,L(16)FIRST
   NAME L(10)NET PAY,S(1)/C1,C2,C3,C126 WHERE C121 EQ 03/30/91
   AND C102 EQ MARKETING:$):
MAP:
```

You could then invoke the string to execute the stored command by simply issuing:

```
*REPORT*
```

Notice that to invoke the string, you place asterisks before and after the string name. (You could also invoke the string with *C1115*.)

The output is shown below:

```
                        MONTHLY REPORT
                          04/01/1991

*           DEPARTMENT-- MARKETING

- EMPLOYEE   LAST NAME            FIRST NAME          NET PAY
   NUMBER
***
-    1007    BROWN                VIRGINA P.          $4,459.79
-    1015    SCHOLL               MADISON A.          $4,308.80
-    1145    JUAREZ               ARMANDO             $4,741.87
-    1050    AMEER                DAVID               $3,180.40
-    1161    RICHARDSON           TRAVIS Z.           $4,634.39
-    1017    WAGGONNER            MERRILEE D.           $962.30
-    1119    GOODSON              ALAN F.             $4,429.10
-    1313    SMITH                JERRY LEE           $2,313.16
-    1234    SHROPSHIRE           LELAND G.           $2,455.17
-    1031    CHAN                 TAI                 $5,846.64
-    1011    VAN HOTTEN           GWENDOLYN           $2,497.70
-    1217    RODRIGUEZ            ROMUALDO R.         $2,544.90
-    1133    WILLIAMSON           JANICE L.           $1,547.64
-    1077    GIBSON               GEORGE J.           $2,468.73
```

Example 2

The software also allows one string to invoke another. For example, you can store the where-clause in the LIST command in Example 1 in a string. Then you can define another string that stores the command and invokes the string with the where-clause, as follows:

```
DEFINE:
1116*QUAL (STRING $C121 EQ 03/30/91 AND C102 EQ MARKETING$):
1117* REPORT1 (STRING $LIST/TITLE D(23)MONTHLY REPORT,B(10),
   B(12)DEPARTMENT--,B(30)
   MARKETING, S(2),L(6)EMPLOY+ NUMBE,B(5)EE+R,L(18)LAST NAME,L(16)FIRST
   NAME L(10)    NET PAY,S(1)/C1,C2,C3,C126 WHERE *QUAL*:):
MAP:
```

You can produce the report by invoking the string containing the string call. (The output would be the same as that shown in Example 1 on the previous page.)

```
*REPORT1*
```

Example 3

The software allows strings to contain parameters or variables, which you indicate by numbers enclosed in asterisks (for example, *1*) in the string definition. When you invoke the string, you supply actual values for these parameters.

For example, in Example 1 for the LIST command, you can use parameters for the name of the department and for PAYROLL MONTH (C121). That string could be defined as follows:

```
1118* REPORT2(STRING $LIST/TITLE D(23)MONTHLY REPORT,B(10),B(12)DEPARTMENT--,
   B(30)*1*, S(2),L(6)EMPLOY+ NUMBE,B(5)EE+R,L(18)LAST NAME,L(16)FIRST NAME,
   L(10)NET PAY,S(1)/C1, C2, C3, C126 WHERE C121 EQ *2* AND
   C102 EQ *1*:$):
```

With string REPORT2, you can produce various reports by simply specifying different values for the parameters. Thus, the report produced by the command in Example 1 can be obtained by invoking string REPORT2 with MARKETING for parameter *1* and 03/30/91 for parameter *2*, as follows:

```
*REPORT2(MARKETING,03/30/91)
```

To obtain the same report for the Information Systems Department, string REPORT2 can be invoked as follows:

```
*REPORT2(INFORMATION SYSTEMS, 03/30/91)
```

The resulting output is shown below.

```
                    MONTHLY REPORT
                      04/01/1991

*           DEPARTMENT-- MARKETING

- EMPLOYEE   LAST NAME          FIRST NAME        NET PAY
   NUMBER
***
-    1043    GIBSON            MOLLY I.          $1,644.35
-    1120    REID              DAVID G.          $1,710.35
-    1067    CAHILL            JACOB             $3,352.29
-    1009    JONES             MICHAEL Y.        $1,838.25
-    1247    GARCIA            FRANCISCO         $1,547.64
-    1266    REDFOX            RICHARD B.        $2,353.66
-    1008    HERNANDEZ         JESSE L.          $3,415.62
-    1123    FREEMAN           LEOPOLD           $2,311.33
-    1049    FERNANDEZ         SOPHIA            $2,613.34
-    1092    POLANSKI          IVAN L.           $3,302.22
-    1010    SMITH             JANET F.          $1,890.85
-    1213    JOHNSON           BRADFORD          $1,526.84
-    1012    QUINTERO          PEDRO             $2,653.40
-    1083    SAVAGE            WILLIAM D.        $1,166.33
-    1139    SEATON            GARY              $1,209.33
-    1156    REED              KENNETH D.        $1,011.70
```

Using Stored Functions

If an ad hoc function is used frequently, you can store it in the database definition either as a string or as a function. However, storing an ad hoc function as a function instead of a string has these two advantages:

- It provides a label (component name) for the function.
- It allows you to specify the data type of the computed result.

Example 4

Example 3 in **Ad Hoc Functions** on page 74 uses ((*FTODAY*-C5)/365.25) to calculate the age of an employee. However, the calculated results would appear without a column heading and in decimal form. You could store that ad hoc function and also assign it a number, a name, and a data type as shown here:

```
DEFINE:
1111*AGE (INTEGER FUNCTION $ ((*FTODAY* - BIRTHDAY)/365.25)$):
MAP:
```

When you invoke the function *AGE* in the LIST command shown below, the calculated values appear under a column heading (AGE) in integer form.

```
LIST EMPLOYEE NUMBER, ((*FTODAY* - BIRTHDAY)/365.25), *AGE* WH C1 LE 1111:
```

```
* EMPLOYEE NUMBER                                AGE
***
*            1043          28.433                 28
*            1001          54.448                 54
*            1003          37.352                 37
*            1002          46.765                 46
*            1101          34.081                 34
*            1006          43.236                 43
*            1004          28.027                 28
*            1007          31.904                 31
*            1062          23.923                 23
*            1015          33.087                 33
*            1050          26.524                 26
*            1017          41.979                 41
*            1031          31.792                 31
*            1011          35.598                 35
*            1077          31.989                 31
*            1074          41.854                 41
      .
      .
      .
```

Example 5

You can use parameters in functions the same way that parameters are used in strings. For example, you can define a function to calculate the GROSS PAY (C124) of an employee who receives a commission.

```
DEFINE:
2222* COMMISSION ( MONEY FUNCTION & (PAY RATE + *1* ) &):
MAP:
```

In this case, the parameter (*1*) is to be used for the amount of commission received by the employee.

You can then use the function, for example, in an update command:

```
ADD C124 = *C2222 (3000.00) WH C121 EQ 05/31/91 AND C1 EQ 1031:
```

Because the software allows strings to invoke functions (and vice versa), the entire command can be stored in a string and the values for C121 and C1 can be made parameters.

```
DEFINE:
2223*COMMISSION ADJUSTMENT (STRING $ ADD C124 = *C2222
   (*1*) WHERE C121 EQ *2* AND C1 EQ *3*:$):
MAP:
```

You can then execute the above ADD command by invoking string C2223 as follows:

```
*C2223 (3000.00,05/31/91,1031)
```

RELOADING A DATABASE

A RELOAD command is available to reload your database.

The RELOAD command reconstructs the files of the attached database. It does an internal unload of existing data and then recreates the database tables. The reload process compacts the database tables and eliminates reusable space. The command requires a considerable amount of scratch file space, equivalent to the amount required to do a load of the database.

The RELOAD command is available in the QUEST language and the CONTROL language. But, after the command is successfully executed, you are given the CONTROL processor.

For information on the RELOAD command, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

Chapter 6

Guidelines for Using the QUEST Language

ENTERING AND EXITING THE SOFTWARE AND THE QUEST PROCESSOR 82
- *Specifying a Password 82*
- *Opening a Database and Calling the QUEST Processor 82*
- *Ending a SYSTEM 2000 Session 82*
- *Changing Processors 83*
  - *Changing processors automatically 83*
  - *Changing processors with commands 83*
  - *Considerations when changing processors 83*

ISSUING QUEST COMMANDS 84

SECURITY CONSIDERATIONS 85

PROCESSING ENVIRONMENTS 86
- *Multi-User Considerations 86*

SYSTEM-WIDE COMMANDS THAT AFFECT THE QUEST LANGUAGE 87
- *System Separator 87*
- *Entry Terminator Word 87*
- *Where-Clause Delimiter 87*

USER FILES 88

MESSAGES 88

ENTERING AND EXITING THE SOFTWARE AND THE QUEST PROCESSOR

This section discusses the commands that are required to open a database, to call the QUEST processor, and to end a session.

Specifying a Password

At the beginning of a SYSTEM 2000 session, the CONTROL processor is automatically attached, and the first command you issue is the USER command, which is a CONTROL language command.

The USER command sets the master password for a new database or verifies the password for an existing database to be opened.

For example,

```
USER, DEMO:
```

Opening a Database and Calling the QUEST Processor

Either the DATA BASE NAME IS command or the RESTORE command opens an existing database, which must have a unique name. These two commands are CONTROL language commands.

For example,

```
DATA BASE NAME IS EMPLOYEE:
```

The DATA BASE NAME IS and RESTORE commands automatically call the QUEST processor. That is, you can issue QUEST commands immediately after issuing either of these commands.

Ending a SYSTEM 2000 Session

To end a SYSTEM 2000 session, issue the following system-wide command:

```
EXIT:
```

Changing Processors

Changing processors automatically The processor lookup capability provides you some flexibility in changing from one processor to another. If processor lookup is activated (with the LOOKUP IS ON command) and you issue a command that is not valid for the current processor, the software will try to switch you to the appropriate processor.

For more information about processor lookup, see **Using Processor Lookup** on page 37-3.

Changing processors with commands You can issue the QUEST, CONTROL, and DEFINE commands to call the QUEST, CONTROL, and DEFINE processors, so that you can issue commands specific to one processor.

Additionally, the following commands automatically call the QUEST processor:

- the CONTROL language DATA BASE NAME IS command
- the CONTROL language RESTORE command
- the DEFINE language MAP command
- the REPORT language GENERATE command.

Considerations when changing processors The commands and options shown below have the following characteristic in common: at the beginning of each session, each one is set to have a certain effect. That effect lasts until changed by the appropriate command or until the session ends.

- Format options affect the results of several commands. Though they can be specified only in the QUEST language, the TREE/RECORD and REPEAT/REPEAT SUPPRESS format options also affect the results of the DEFINE language MAP command.
- Limits set with the QUEST language LIMIT command can affect the results of the DEFINE language MAP command when it includes a where-clause.
- You can use the ENABLE/DISABLE EXECUTION commands in QUEST and in DEFINE. They affect command execution in both processors.
- The QUEST language SAME IS command affects the processing of the SAME operator.

ISSUING QUEST COMMANDS

The following general considerations apply to QUEST commands:

- You can begin or end commands in any column.
- Spacing is free-form; that is, commands can have any number of blanks between words. Extraneous blanks (leading, trailing, or more than one embedded blank) are ignored except in TEXT and UNDEFINED values and LIST command layout specifications.
- You can continue commands from one line to the next.
- You can enter more than one command on a line. The only exceptions are the four system-wide commands used to specify an alternate user file. Any of these commands must be either the only command on a line or the last one.
- Each command must end with a command terminator. The default command terminator is the colon.
- Component names and numbers are interchangeable, except in value streams and the Data File, where you use only the component number (without the preceding C).

  When you use component numbers (C-numbers), the software accepts a C followed by a maximum of four digits. The software treats the numeric characters as a number; therefore, leading zeroes are acceptable. For example, PRINT C2 and PRINT C0002 are both acceptable.

As most of these rules indicate, you can issue the same command in many ways. The following examples show different ways of issuing the same QUEST command.

```
PRINT EMPLOYEE NUMBER WHERE LAST NAME EQ SMITH:  1

PR C1 WH C2 EQ SMITH:  2

PRINT C1 WHERE  3
         C2 EQ SMITH:

        PR C1  WH   C2 EQ         SMITH:  4
```

1 This example includes component names.

2 This example includes component numbers and abbreviations for keywords.

3 This example shows continuing a command from one line to the next.

4 Commands can begin or end on any line and spacing is free-form. Because C2 is not a TEXT item, the extra blanks between EQ and SMITH are ignored when the command is processed.

SECURITY CONSIDERATIONS

SYSTEM 2000 software provides security based on passwords. The master password holder has unqualified access to the database. The DBA password holder can perform administrative functions but cannot access data. For a DBA password holder to issue certain commands, the master password holder must assign authority for specific commands to the DBA password.

Secondary passwords can be assigned with authorities granted for individual components. For any particular combination of components, a secondary password holder can be granted retrieval authority, update authority, and/or qualification authority.

- To issue a retrieval command, a secondary password holder must have retrieval authority (R) for every component specified in the retrieval-clause and ordering-clause.

- To issue an update command, a secondary password holder must have update authority (U) for every component specified in the action-clause.

- To issue a command with a where-clause, a secondary password holder must have qualification authority (W) for every component specified in the where-clause.

- To invoke a stored string or stored function, a secondary password holder must have authority to use the string or function and must also have the appropriate authority for each component in the string or function.

The master password holder assigns passwords, secondary password authorities, and DBA password command authorities with the CONTROL language. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information on passwords.

PROCESSING ENVIRONMENTS

You can use the QUEST language in either batch or interactive mode. And, you can use the QUEST language in a single-user or Multi-User environment. In a single-user job, you have your own copy of SYSTEM 2000 software. In a Multi-User environment, multiple users share a single copy. This includes users submitting jobs in either batch or interactive mode. The Multi-User software ensures data and structural integrity, allowing multiple users to concurrently retrieve and update a database.

For information on the Job Control Language (JCL) to run in a single-user or Multi-User environment, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

Multi-User Considerations

In a Multi-User environment, two or more users share a single copy of SYSTEM 2000 software in a single region in main memory. A single copy of the software can support several teleprocessing (TP) monitor systems.

An important concept when using the software interactively in a Multi-User environment is that of a *TP segment*. A TP segment begins with the first character transmitted and ends with a message completion signal for the hardware, such as the Enter key, carriage return, or EOT (end of transmission).

A TP segment consists of one or more commands in any Self-Contained Facility language. (One SYSTEM 2000 thread is tied up for the duration of a segment.)

Considerations for a TP segment are as follows:

- One or more commands must be completed in a TP segment. Otherwise, the following message appears:

```
-803- UNEXPECTED END OF COMMAND FILE -
```

- Each TP segment can contain up to 1200 characters. If that limit is exceeded, message -803- appears.

- If the DITTO execution parameter equals YES, you can use the DITTO operator across TP segments. If DITTO equals NO, the DITTO operator can refer only to an action-clause specified in a command in the current segment. Otherwise, the following message appears:

```
-302- NO PRIOR VALID COMMAND -
```

- If the SAME execution parameter equals YES, the SAME operator can refer to where-clauses across TP segments. If SAME equals NO, the SAME operator can refer only to a where-clause specified in a command in the current TP segment. Otherwise, the following message appears:

```
-367- NO PRIOR VALID WHERE-CLAUSE -
```

- The PREVIOUS operator can refer only to a value or value stream specified in a command in the current TP segment. Otherwise, the following message appears:

```
-359- NO PRIOR ACCEPTABLE VALUE STRING -
```

- The Command File can be switched to an alternate file in a TP segment. The COMMAND FILE IS command specifying the alternate file must be the last command in the segment, or results are unpredictable. The DD statement for the alternate file must be in the Multi-User JCL.

For more information about TP segments and the SAME and DITTO execution parameters, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

SYSTEM-WIDE COMMANDS THAT AFFECT THE QUEST LANGUAGE

There are three system-wide commands that change the special characters or words used in many QUEST retrieval or update commands. For more information about these and other system-wide commands, see the **SYSTEM-WIDE COMMANDS** part in this manual.

System Separator

The system separator is a signal to the software, for example, to show the end of a value. You use the system separator in the ADD, ASSIGN, ASSIGN TREE, CHANGE, and INSERT TREE commands.

The default system separator is the asterisk. You can change it with the SEPARATOR IS command.

Entry Terminator Word

The entry terminator word marks the end of a value stream. You use the entry terminator word in the ADD, ASSIGN, ASSIGN TREE, CHANGE, and INSERT TREE commands.

The default entry terminator word is END. You can change it with the ENTRY TERMINATOR IS command.

Where-Clause Delimiter

The where-clause delimiter is used only in QUEST language where-clauses.

The default where-clause delimiter is the slash. You can change it with the DELIMITER IS command.

USER FILES

There are four user files for communications between you and SYSTEM 2000 software.

- The Command File is an input file from which the software reads your commands.
- The Data File is an input file from which the software reads values used in executing the LOAD command.
- The Report File is an output file on which the software writes the results of retrieval commands.
- The Message File is an output file on which the software writes error messages and informative messages.

The two input files, by default, are named INPUT. INPUT identifies the device (such as your terminal) from which the software reads either commands or input values. The two output files, by default, are named OUTPUT. OUTPUT identifies the device (such as your terminal or printer) on which the software writes messages and the results of retrieval commands.

You can specify alternate user files by issuing these commands: COMMAND FILE IS, DATA FILE IS, MESSAGE FILE IS, and REPORT FILE IS. For information on these system-wide commands, see the **SYSTEM-WIDE COMMANDS** part in this manual.

MESSAGES

During a SYSTEM 2000 session, messages appear as necessary in response to your commands. There are three general types of Self-Contained Facility messages:

- **Informative**
  Informative messages appear during normal processing, informing you of procedures undertaken and completed by the software. Informative messages are not fatal to the job and are never destructive to a database.
- **Warning**
  Warning messages tell you that the software has executed a syntactically acceptable command that could have undesirable consequences. Or a warning can indicate that the database was damaged before the current command was issued. If the current command is a request to update the database, the update is not allowed.
- **Error**
  Error messages notify you that a command contains an error.

For more information about messages that can appear during a SYSTEM 2000 session, see *SYSTEM 2000 Messages and Codes, Version 12, First Edition.*

In addition, Appendix A in this manual contains information on how to avoid errors.

REFERENCE

Loading Data

Choosing a Method for Loading a Database

Using the LOAD Command

Chapter 7

Choosing a Method for Loading a Database

METHODS FOR LOADING 7-1
INSERT TREE Command 7-2
LOAD Command 7-3
PLEX 7-3

SELECTING A METHOD 7-3
Factor I. Amount of Data 7-4
Factor II. Number of Times Data are to be Added 7-4
Factor III. Source of Data 7-5
Factor IV. Logical Position of New Entries 7-6
Factor V. Physical Position of New Entries 7-6
Factor VI. Error Handling 7-7
Factor VII. Resources 7-7

METHODS FOR LOADING

After you define a SYSTEM 2000 database, it is ready to be loaded. You have several ways to load data:

- Use the QUEST language INSERT TREE command.
- Use the QUEST language LOAD command.
- Use the SYSTEM 2000 Programming Language Extension (PLEX).

These methods are described in the following sections.

INSERT TREE Command

Use the INSERT TREE command to load one logical entry at a time. That is, issue one INSERT TREE command for each entry to be loaded. The INSERT TREE command allows you to specify the position at which each new entry is to be placed.

For example, the following command uses trace notation to insert a logical entry in position 13.

```
INSERT TREE C0 *13 EQ value-stream:
```

The following command uses an after-clause to insert a logical entry after the entry for the employee with C1 EQ 1010:

```
INSERT TREE C0 EQ value-stream AFTER C1 EQ 1010:
```

You can also include INSERT TREE commands within a parametric string. Some or all of the values for the logical entry can then be specified when the string is invoked. It is also possible to define a string so that trace notation and the value stream can be specified when the string is invoked.

For example, you can invoke the following string to place an entry in any position:

```
2001*UPDATE (STRING $ INSERT TREE C0 **1* EQ *2*:$):
```

Then, the following command enters the employee's data as the 32nd logical entry.

```
*C2001 (32, 1*1032*2*BROWN*3*...*END*):
```

For information on the INSERT TREE command, see **Adding New Data Trees: INSERT TREE** on page 21-15.

LOAD Command

You can use the QUEST language LOAD command to enter one or more logical entries into a database. To use this command, the entries must be in a file in loader stream format. The order in which the logical entries appear in the file determines the order in which they are loaded into the database. If the database already contains some entries, new entries are placed after the original ones.

To load the data, identify the file containing the data with the system-wide command DATA FILE IS, and then issue the LOAD command. Those entries that contain no errors are loaded into the database. If there are errors, you can correct them, then load those data.

For information about the LOAD command, see Chapter 8, "Using the LOAD Command."

PLEX

To load a database using a PLEX program, the data must reside on a file (disk or tape). You create a load program written in either Assembler, COBOL, FORTRAN, or PL/I. A load program looks like a standard program written in any of these languages, with PLEX commands embedded.

For information about PLEX, see the *SYSTEM 2000 PLEX Manual, Version 12, First Edition.*

SELECTING A METHOD

This section outlines the main factors in choosing a method to load a database. For each factor, one or more cases are considered and suitable methods are suggested.

Before you decide on a method, consider these and other factors as they apply to your database. In some cases, more than one method is appropriate. Note that each factor is not a stand-alone topic; do not make a decision based on only one factor without considering the consequences. For example, the format of the data to be loaded may suggest using the LOAD command. On the other hand, the need for the fastest run-time may suggest using PLEX load mode. You must take all factors under consideration before deciding on any one method.

Factor I. Amount of Data

Consider the amount of data to be entered when the database is created (that is, the initial load).

Case A. Small quantity of data.

1. If the data are available in loader stream format, use the LOAD command.
2. If the data are not available in loader stream format,
   a) enter the data with the INSERT TREE command
   b) or build a file with the data in loader stream format and use the LOAD command.

Case B. Large quantity of data.

1. If the data are available in loader stream format, use the LOAD command.
2. If the data are not available in loader stream format,
   a) write a data-editing program and a PLEX load program. (The editing program builds the input file for the PLEX load program.)
   b) or build a file with the data in loader stream format and use the LOAD command.

Factor II. Number of Times Data are to be Added

Consider the number of times entries are to be added to the database during its life span.

Case A. The database is to be loaded only once. (New entries will not be added to the database once it is loaded.) Use the LOAD command or write a PLEX load program, depending on the size of the database. See Factor I.

Case B. After the database is initially loaded, only a few entries will be added to it. Use the LOAD command, or write a PLEX load program for the initial load. Use INSERT TREE commands for subsequent new entries.

Case C. The database is to have entries added to it periodically.

1. If large quantities of data are to be added each time, write a PLEX load program.
2. If small quantities of data are to be added each time,
   a) use a PLEX program initially, and also use that program for updating
   b) or, if you use the LOAD command initially, also use it for updating.

Factor III. Source of Data

Consider the source of the data.

Case A. Data in machine-readable form.

1. Input data reside on one file (for example, disk or tape).

   a) If the file contains output of an UNLOAD command, the data are already in loader stream format, so you can use the LOAD command.

   b) If the file was created by an existing system (not with the UNLOAD command), write a PLEX load program.

2. Input data reside on several files.

   a) Each file contains data for separate groups of entries. For example, File 1 contains data for employees in the Marketing Department, File 2 contains data for employees in the Information Systems Department, and so on.

      - If each file contains the output of an UNLOAD command, identify each file as a Data File, then use the LOAD command.

      - Otherwise, write a PLEX load program.

   b) Each file contains data for one or more records for each logical entry. For example, File 1 contains C0 data, File 2 contains salary data, and so on.

      Write a merge program to produce a file to be read by a PLEX load program.

3. Input data reside on one or more databases.

   a) Use UNLOAD commands to retrieve data from each database, and place the retrieved data in one or more files. Use these files as input files to load a database with the LOAD command. This method applies only if component numbers, item types, and pictures agree in the databases.

   b) Write a PLEX program that retrieves data from several databases and then inserts the data directly into another database. This method applies even if the component numbers do not agree; however, item types and pictures must be compatible.

4. Input data require manipulation. For example, data are embedded in one or more files or some data need conversion (for example, dates are in a format that is not acceptable to the software).

   Write a program that builds a data file obtained from the other file(s). Use that file as input to a PLEX load program.

Case B. Data not in machine-readable form.

Collect data from source documents to build a file in loader stream format for use with the LOAD command or in some other format to be used with a PLEX load program.

Factor IV. Logical Position of New Entries

Consider the logical position of new entries to be added to an existing database.

Case A. New entries to be placed after existing entries. Use an appropriate method. See Factor II.

Case B. New entries to be placed in specific positions among existing entries.

Write a PLEX load program or use INSERT TREE commands. Depending on the number of entries to be inserted at any given position, the PLEX program insertions could be in immediate, queue, or load mode. For few insertions, use immediate mode; for many insertions, use load mode.

Factor V. Physical Position of New Entries

Consider the physical position of records in new entries to be added to an existing database.

Case A. LOAD and INSERT TREE commands place new data records in reusable space in the Hierarchical Table and in the Data Table, where possible. Also, reusable space is used when loading a database using PLEX immediate mode or PLEX queue mode.

Case B. PLEX load mode uses available space only, rather than reusable space.

Factor VI. Error Handling

Consider error handling for the database.

Case A. Error checking can be done by an editing program and by a PLEX load program. It can be as complex as necessary to validate the contents and placement of each data record in the input file. With a large volume of data, automated error-checking may be essential. The more sophisticated the program's error checking, the more likely that only valid items and records are loaded.

Case B. Using the LOAD command, you rely on SYSTEM 2000 error-checking as well as visual examination. Nevertheless, there is still the risk of creating misplaced records or some other error that could be prevented by writing a validation program.

Factor VII. Resources

Consider your resources.

Case A. Skills.

1. Writing a PLEX load program requires a knowledge of Assembler, COBOL, FORTRAN, or PL/I, as well as PLEX.

2. The LOAD command does not require any programming knowledge.

Case B. Time.

1. You can use the LOAD command as soon as a database is defined. However, preparing a file in loader stream format can be time-consuming.

2. Writing a PLEX load program requires design, coding, and testing time. However, once the program is written, you can use it as often as needed. Also, depending on the source of the data (see Factor III), no time is required to prepare a Data File.

The following table compares the methods of loading a database according to five factors:

- run time
- resulting structure
- usage of reusable and scratch file space
- effect of commands on the Update Log.

Illus. 7.1 Comparison of Methods to Load a Database

	Run Time	Resulting Structure	File Space Usage	Effect on Update Log
INSERT TREE	Slowest - Index updated for each command.	Records might be scattered if reusable space is used.	Least usage.	Commands are logged.
LOAD	Medium - HT, DT, and index data on scratch files. Index updated once.	Records might be scattered if reusable space is used.	Most usage.	Command logged.
PLEX load mode	Fastest - HT and DT updated immediately after each insert. Index updated after TERMINATE.	Records in HT and DT are contiguous.	Medium usage.	Update Log cancelled.

HT = Hierarchical Table
DT = Data Table

Chapter 8

Using the LOAD Command

INTRODUCTION 8-2

THE LOAD COMMAND 8-3
- *Format 8-3*
- *Example: Load Messages 8-4*
- *Steps In Loading 8-5*
- *Loading Techniques 8-6*
- *Loading a Database Interactively 8-8*

PREPARING THE DATA FILE 8-10
- *Organizing Entries in the Data File 8-10*
- *Organizing Data Records In a Logical Entry 8-11*
- *Coding Values 8-12*
  - *Coding INTEGER values 8-13*
  - *Coding DECIMAL values 8-13*
  - *Coding MONEY values 8-14*
  - *Coding REAL and DOUBLE values 8-14*
  - *Coding dates 8-15*
  - *Coding CHARACTER values 8-16*
  - *Coding TEXT values 8-16*
  - *Coding UNDEFINED values 8-17*
- *Efficiency Considerations 8-18*
  - *Criteria for sequencing logical entries 8-18*
  - *Loading null records 8-18*

TYPICAL PROBLEMS WITH LOADING 8-19

THE LOADING PROCESS 8-24
- *Data Scanning 8-24*
- *Updating the Database Tables 8-25*

This chapter describes the LOAD command, which loads a database.

INTRODUCTION

The LOAD command enters new logical entries into a database. That is, it directs the software to read a Data File containing values for each logical entry and to store the data in the database tables. The software then gives you a report on the success or failure of the loading process.

The loading process enters values and the structure for new entries into the database; it does not add values or data records to existing entries. You can load one entry or a batch of new entries as often as desired. For example, you can enter source data for this year and immediately begin to use those data. Then, you can enter a batch of last year's data and so on until all source data from a backlog of files have been put into the database.

The load process uses all padding options in the current definition and also uses all space in the database that is available due to REMOVE TREE operations.

The source data need not be complete, nor need they be totally accurate or standardized. Update requests can, at some later time, modify or insert any piece of data individually or all the data across the database.

A database schema must exist before you can load values into a database. The only commands that are mandatory for initiating the loading process are the DATA FILE IS command and the LOAD command.

THE LOAD COMMAND

You can enter either one logical entry or multiple entries with the LOAD command. After you issue the command, the software reads values from a specified Data File.

Format

```
LOAD:
```

Before you issue the LOAD command:

- The data to be loaded must be written to a Data File in loader stream format.
- You must allocate the Data File with the appropriate operating system statement.
- You must identify the file containing the data with the DATA FILE IS command.
- You can also issue the DISABLE EXECUTION command or the STOP AFTER SCAN IF command. Regardless of which command you issue, the software scans the Data File for errors during the load process. The STOP AFTER SCAN IF command prevents loading if any error is found; the DISABLE EXECUTION command prevents loading even if no errors are found. (For information on these commands, see **Controlling Execution in Event of Errors: STOP/CONTINUE** on page 31-15 and **Preventing Execution of Commands: ENABLE/DISABLE EXECUTION** on page 31-18.)

  If you direct the loading process to stop after scanning the data for any reason or to stop after errors and errors do indeed occur, the database is not altered. That is, scanning only does not affect the database.

The only output from the loading process is the valued database reflecting the accepted entries. If errors occur, the Message File contains the pertinent error messages. The database cycle number is incremented by one each time the loading process is completed, but the loading process does not affect the definition number.

To protect against damage when the load process is complete, the software clears all database pages from memory and writes them to disk.

Example: Load Messages

An example of the messages displayed after the loading process is shown here. During the loading process, the software issues output lines at various stages of execution. Each line begins with a message number and a 24-hour clock time of *hh:mm:ss*, where *hh* = hour, *mm* = minutes, and *ss* = seconds.

```
LOAD:

-241-hh:mm:ss - BEGIN LOADING -

-239-hh:mm:ss - SCANNING VALUE NO. <n> -
                    .
                    .
                    .
-240-hh:mm:ss - SCAN COMPLETE -

-242-hh:mm:ss - HIERARCHICAL STRUCTURE
     UPDATING COMPLETE -

-243-hh:mm:ss - BEGIN SORT OF RECORD
     UPDATES -

-244-hh:mm:ss - SORT OF RECORD UPDATES
     COMPLETE -

-245-hh:mm:ss - RECORD UPDATING
     COMPLETE -

-246-hh:mm:ss - BEGIN KEY VALUE SORT -

-247-hh:mm:ss - KEY VALUE SORT COMPLETE -

-248-hh:mm:ss - BEGIN LONG VALUE SORT -

-249-hh:mm:ss - LONG VALUE SORT COMPLETE -

-250-hh:mm:ss - LONG VALUE UPDATE
     COMPLETE -

-251-hh:mm:ss - LOADING COMPLETE -
```

Some of these messages do not necessarily display for every load.

- Message -239- is issued after every 10,000 values are scanned.
- Messages -243- and -244- are issued only when reusable space was used in the session.
- Messages -246- and -247- are issued only when key values were sorted.
- Messages -248-, -249-, and -250- are issued only when a value for CHARACTER or TEXT items is longer than the picture.

Steps In Loading

To load a database, follow these steps, in the order given. (These steps assume that the database has been defined and that space has been allocated for the database files.)

1. Prepare the Data File. The Data File can be one or more files containing the data to be loaded. The entries must be in loader stream format corresponding to the database schema.

2. Allocate the Data File with the appropriate operating system statement.

3. Issue the DATA FILE IS command.

4. Call the QUEST processor, if needed, and issue either the DISABLE EXECUTION command or the STOP AFTER SCAN IF command.

5. Issue the LOAD command. The software scans the file for errors and issues appropriate messages.

6. If there are no errors, and you issued the STOP AFTER SCAN IF command, the database is loaded, and you can issue any QUEST language command or system-wide command. If you are loading the database by increments, repeat steps 3 through 5 to load another group of entries.

7. If there are no errors and you issued the DISABLE EXECUTION command, you must issue the commands given below in order to load the data. (Assume that file MYDATA contains the data to be loaded. The command DATA FILE IS INPUT repositions the MYDATA file.)

```
DATA FILE IS INPUT;
DATA FILE IS MYDATA;
ENABLE EXECUTION;
LOAD;
```

8. If there are errors, you must edit the Data File or modify the database definition.

9. Once the database is loaded, if the database is new and the items were defined as non-key, you can use CREATE INDEX to make the appropriate items key items. If the database is old, use REORGANIZE to reorganize the index. Only the master password holder can issue the CREATE INDEX command. The master password holder or the DBA password holder can issue the REORGANIZE ALL command.

Loading Techniques

Depending on the volume of data and the availability of machine time, you can load a database using one or more of the following techniques:

- Load the data with one LOAD command.
- Load the data in increments. That is, instead of loading all entries at once, divide them into groups and use a LOAD command to load each group.

  After the last group is loaded, the master password holder or DBA password holder can issue the CONTROL language command REORGANIZE to organize the index as though all the entries had been loaded at one time. (For information on the REORGANIZE command, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.)
- Change the database schema so that it contains only non-key items and load the data, either incrementally or all at once.

  After the load, the master password holder can issue the CONTROL language CREATE INDEX command to change non-key items to key (as necessary), which builds the index for these items. For information on the CREATE INDEX command, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

Ideally, a single load is better than multiple loads. A single load provides better organization of the database tables, especially of the index, thereby lowering the number of I/Os that will be required for later retrievals and updates. However, a single load also involves a costly run in terms of scratch file space and execution time. Also, these two factors carry a higher risk of your having to redo the load process due to machine failure.

By using incremental loads, followed by either a REORGANIZE INDEX command or a CREATE INDEX command, you avoid large scratch file requirements while still organizing the index as though a single load had been used.

The following example shows how to load database SAMPLE with incremental loads. Assume that the data reside on three files: LOAD1, LOAD2, and LOAD3. Notice that no errors are found in the process.

```
USER, NEW:
---
DATA BASE NAME IS SAMPLE:
---
 -556- OPENED.....SAMPLE                1      0 05/22/1990  18:07:59
---
STOP AFTER SCAN IF ANY COMMAND IS REJECTED:
---
DATA FILE IS LOAD1:
---
LOAD:
 -241- 18:08:05 - BEGIN LOADING -
 -240- 18:08:05 - SCAN COMPLETE -
 -242- 18:08:07 - HIERARCHICAL STRUCTURE UPDATING COMPLETE -
 -245- 18:08:07 - RECORD UPDATING COMPLETE -
 -246- 18:08:07 - BEGIN KEY VALUE SORT -
 -247- 18:08:07 - KEY VALUE SORT COMPLETE -
 -251- 18:08:07 - LOADING COMPLETE -
---
DATA FILE IS LOAD2:
---
LOAD:
 -241- 18:08:08 - BEGIN LOADING -
 -240- 18:08:09 - SCAN COMPLETE -
 -242- 18:08:09 - HIERARCHICAL STRUCTURE UPDATING COMPLETE -
 -245- 18:08:09 - RECORD UPDATING COMPLETE -
 -246- 18:08:09 - BEGIN KEY VALUE SORT -
 -247- 18:08:09 - KEY VALUE SORT COMPLETE -
 -251- 18:08:09 - LOADING COMPLETE -
---
DATA FILE IS LOAD3:
---
LOAD:
 -241- 18:08:10 - BEGIN LOADING -
 -240- 18:08:10 - SCAN COMPLETE -
 -242- 18:08:10 - HIERARCHICAL STRUCTURE UPDATING COMPLETE -
 -245- 18:08:10 - RECORD UPDATING COMPLETE -
 -246- 18:08:10 - BEGIN KEY VALUE SORT -
 -247- 18:08:10 - KEY VALUE SORT COMPLETE -
 -251- 18:08:13 - LOADING COMPLETE -
---
CONTROL:
---
REORGANIZE ALL:
---
    .
    .
    .
```

Loading a Database Interactively

SYSTEM 2000 software does not allow the Command File and the Data File to be the same physical file. If you want to enter the loader stream interactively, this restriction can be circumvented for single-user jobs by using the procedure presented in this section.

The main idea in this procedure is to make file INPUT, which identifies your terminal, be the Data File. To do this, specify an alternate Command File. (In this example, the alternate Command File is named OTHER. It is allocated in the single-user JCL with the DDname of OTHER.) The file contains one record with two commands:

```
LOAD:  COMMAND FILE IS INPUT:
```

When you begin a session, the Command File is INPUT by default. You can submit the following command to tell the software to read the next commands from file OTHER:

```
COMMAND FILE IS OTHER:
```

At this point, the Command File is no longer INPUT and the software can use file INPUT as the Data File. The software reads the first command in OTHER, which is

```
LOAD:
```

The LOAD command causes the terminal to be prompted for input. The software expects data to be entered in loader stream format. At this point, the current Command File is OTHER and the Data File is INPUT, that is, your terminal. This remains true until the software finds the end of the Data File, which is two consecutive system separators, like **. Or they could have blanks between them, like * *.

When the software finds the two asterisks, it completes the loading process and enters the values into the database tables. When the load process is finished, the software returns to the current Command File OTHER for the next command and finds the following command:

```
COMMAND FILE IS INPUT:
```

This command redirects the software to the original input file INPUT (your terminal) for subsequent commands.

The next example shows this process. This example assumes that file OTHER has been allocated and that you are ready to load the data. Lines that you would enter are preceded by an arrow.

```
  ---
→ COMMAND FILE IS OTHER:
  ---
  LOAD: COMMAND FILE IS INPUT:        (echo of commands in file OTHER)
  ---
   -241- 18:06:13 - BEGIN LOADING -
→ 1*1014*2*BROWN* ... *END*
     .
     .
     .
→ 1*1267*2*JOHNSON* ... *END*
→ *                                   (end of loader stream)
   -241- 18:06:47 - SCAN COMPLETE -
     .
     .
     .
   -251- 18:07:36 - LOADING COMPLETE -
  ---
→ PRINT COUNT C0:
  ---
     .
     .
     .
```

PREPARING THE DATA FILE

A Data File is composed of values organized according to the database schema. Each value is preceded by an item number, and each group of values (data record) is preceded by a schema record number. The item numbers and record numbers are not preceded by the letter C.

First, organize your data by logical entry. Next, arrange the values within each entry according to the schema records. There is no maximum number of data records at any level.

The end of a value is indicated by a system separator, and the end of a data record is indicated by either another schema record number or an entry terminator (END*), which designates the end of a logical entry. If the software finds an entry terminator, the next item number and value found in the Data File begin the next logical entry.

The end of the Data File is signaled by one or more system separators after the last entry terminator. Blanks before, after, and between separators are not necessary; they are ignored if found. The following examples are all acceptable ways to end a Data File:

```
END *      *
END        **
END**
END   *    *
```

The end of a Data File means the end of the data for a particular LOAD command. Sometimes it is useful to place the final separator on a line by itself. This allows flexibility in rearranging the entries before signaling the end of a Data File.

In composing the Data File, you form an implied data tree of data records and also an implied order of entrance into the database. Each new entry is loaded at the end of the database.

Organizing Entries in the Data File

For the loading process, the Data File is a group of logical entries. The order of the entries is the order in which they are loaded into the database.

The loading process stops immediately if two consecutive asterisks occur before the end of the last data entry, which signals the end of the Data File.

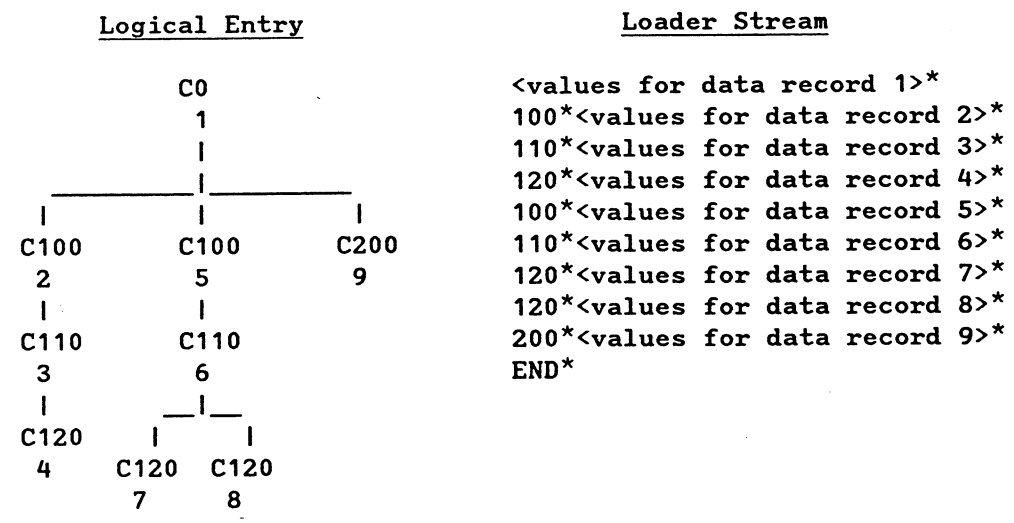
Organizing Data Records In a Logical Entry

Each logical entry in a Data File is a hierarchically ordered set of data records. The data records are placed in logical order; therefore, the C0 record is always placed first. The C0 record is formed by listing the schema item numbers and their associated values; schema record number 0 is not used.

Each non-level 0 data record is formed by your giving the schema record number, the system separator, and values (identified by item numbers) for that particular data record. Thus, any schema record number found in the loader stream signals the end of one data record and the beginning of the next. All items assigned data values in a data record must belong to the specified schema record. You must give the values for all items in each data record together in the loader stream before you go to another data record. However, the order of the items within a data record is immaterial, as they will be stored in schema order.

The following illustration shows the format of a loader stream for one logical entry:

Illus. 8.1 Loader Stream Format for a Logical Entry



A data record can contain only one value for each item in a schema record. If you do not have a value for a particular item, omit the item number from that data record. Thus, a data record can contain values for all items in a schema record, for a few of the items, or perhaps for none of the items.

If no values exist for a data record, that data record is empty (null). Generally speaking, a null data record will not occur unless you have source data for descendant data records.

You must give all level 0 values together at the beginning of each logical entry. If no values exist for any of the level 0 items, the scanner automatically creates an empty level 0 data record so that descendant data records can be associated properly for that logical entry.

When you enter values at any level below a data record having no values, you must specify an empty data record in the loader stream before entering its descendants. To do so, specify the component number for the parent schema record, followed by a system separator and the component number for the descendant record.

The software retains the original order in which data records were entered in the database. In Illus. 8.1 on page 8-11, if data record 9 had been loaded before data record 5, the data record would show record 9 to the left of record 5. A PRINT ENTRY command would print record 9 data after all descendants of record 2 and before record 5. (The ordinal position of data records is true for any database, but it is not significant to you unless you want to use it in retrievals or updates.)

Coding Values

You must use the system separator after each component number and after each value. If the system separator will occur in a data value, you must change the system separator before issuing the LOAD command. (See Chapter 40, "Changing the System Separator: SEPARATOR IS.")

Blanks can occur before and after the item number and after the separator. All blanks are ignored except blanks that are to be considered as part of a TEXT or UNDEFINED value.

The loader stream for Logical Entry Two of the EMPLOYEE database (see fold-out) is shown below. The loader stream contains data for records 12 through 23.

Illus. 8.2 Example of a Loader Stream

```
1*1120*2*REID*3*DAVID
G.*4*02/16/1988*5*08/15/1957*6*441-04-0121*7*MALE*
8*CAUCASIAN*9*FULL TIME*10*410
XT369*11*80.00*12*80.00*13*224*
14*1302 LAZY LANE*15*AUSTIN TX*16*78752*
100*101*PROGRAMMER*102*INFORMATION SYSTEMS*103*MYJ*104*PROFESSIONAL*
105*03/01/1989*
110*111*1800.00*112*MONTHLY*113*03/01/1990*
110*111*1650.00*112*MONTHLY*113*03/01/1989*
110*101*ASSISTANT PROGRAMMER*102*INFORMATION SYSTEMS*103*MYJ*
104*PROFESSIONAL*105*02/16/1988*106*02/28/1989*110*111*$1500.0*
112*MONTHLY*113*02/16/88*
200*201*PL/I*202*EXCELLENT*203*5*
200*201*COBOL*202*GOOD*203*1*
300*301*CHESS*302*AUSTIN CHESS CLUB*
400*410*411*SOUTHERN METHODIST UNIVERSITY*412*BS*413*12/16/1988*
414*ELECTRICAL ENGINEERING*415*COMPUTER SCIENCE*
420*421*TECNIK*422*SYSTEM RS ADVANCED*423*03/18/1989*END*
```

Coding INTEGER values When loading INTEGER values, an error occurs if the length of the value exceeds the picture. All characters in the picture can be used as numeric digits. Because numeric values are stored in packed decimal format, a plus or minus sign is associated with each value. You can also use E-notation in the form of ±n.nE±ee.

Loading values for an INTEGER item having a picture of 9999 gives the following results:

Value Loaded for Item C1	Value Displayed with PRINT C1
987	987
+1456	1456
12345	error - exceeds picture width
-1456	-1456
-ß1456	-1456
-ßß1456	-1456
14.56	error - value is not an INTEGER
10.998E00	11

Coding DECIMAL values When loading DECIMAL numbers, the values need not have decimal points since the picture controls the decimal placement. All characters in the picture can be used as numeric digits. Values can be loaded with a minus sign indicating negative numbers. Blanks can (but need not) occur between the sign and the numerals. Commas can be included for numbers in the thousands; they are ignored. You can also use E-notation in the form of ±n.nE±ee.

When loading DECIMAL values, an error occurs if the number of digits to the left of the decimal point exceeds the number for that position specified in the picture. If the number of digits to the right of the decimal point exceeds the number specified in the picture, the value is truncated.

Loading values for a DECIMAL item with a picture of 99.99 gives the following results:

Value Loaded for Item C1	Value Displayed with PRINT C1
23.45	23.45
+23.45	23.45
00023.45	-error- exceeds picture width
+ß23.45	23.45
+ßß23.45	23.45
2345	23.45
23450	-error- exceeds picture width
-23.45	-23.45
-ß23.45	-23.45
23.451	23.45 (truncates)
23451	-error- value exceeds picture width
23	.23
-23	-0.23
123.45	-error- value exceeds picture width
10.998E00	11.00

Coding MONEY values MONEY values follow the same rules as DECIMAL values, except for the optional use of a dollar sign ($) and a CR (credit) for negative values. That is, a dollar sign can precede the value. Blanks can (but need not) occur between the $ and the numerals. If a sign (+ or -) is used, it can be given before or after the dollar sign, with or without a single trailing blank. You can indicate negative values either by using a minus sign or by using CR to the right of the value. You can also use E-notation in the form of ±n.nE±ee.

Loading values for a MONEY item with a picture of $9999.99 gives the following results:

Value Loaded for Item C1	Value Displayed with PRINT C1
23.45	$23.45
$23.45	$23.45
$b23.45	$23.45
+$23.45	$23.45
$+23.45	$23.45
$b+23.45	$23.45
$b+b23.45	$23.45
+b$b23.45	$23.45
7923.45	$7,923.45
7,923.45	$7,923.45
792345	$7,923.45
- 92345	$923.45CR
$b-2345	23.45CR
$b23.45CR	$23.45CR
23.45CR	$23.45CR
10.998E00	$11.00

Coding REAL and DOUBLE values REAL and DOUBLE items contain floating point values. REAL means single precision numbers; DOUBLE means double precision numbers. You do not define a picture for REAL or DOUBLE items. Each REAL value occupies one word (four bytes) in the database. Each DOUBLE value occupies two words (eight bytes). Overflow is not allowed.

Numeric data for scientific calculations are commonly represented in floating point. A floating point value consists of two parts: a mantissa giving the value and an exponent giving the value's magnitude. The exponent for E-notation values must be in the range of $-79 < E < +75$. You can also enter whole numbers or numbers containing a decimal point (not E-notation) as input for REAL and DOUBLE items. Negative zero is allowed as a value, but it is accepted as positive zero.

When you use ADD, CHANGE, or ASSIGN to update REAL and DOUBLE values, truncation or padding occurs as needed. No messages appear.

Note: the output format for REAL and DOUBLE values is E-notation, that is, ±n.nE±ee, where n.n is a mantissa giving the value and ee is an exponent giving the value's magnitude. The word -NULL- is displayed for a missing value.

Coding dates The default format for dates is MM/DD/YY, where MM is a two-digit month code, DD is a two-digit day code, and YY is a two-digit year code. For the year, the software assumes that the century is the current one; that is, the prefix is assumed to be 19. You can change the date format by issuing the DATE FORMAT IS command. See Chapter 27, "Changing the Date Format: DATE FORMAT IS."

When loading dates, an error occurs in the following situations:

- The date does not correspond to the date format in effect.
- The date is invalid.
- The value is before October 15, 1582, or after December 31, 2499.
- An invalid character appears.

Loading values for a date with the format of MM/DD/YYYY gives the following results:

Value Entered	Value Displayed
11/23/1876	11/23/1876
11/23/88	11/23/1988
01/01/01	01/01/1901
02/3/1989	error - incorrect format for day
02/30/1989	error - invalid date
JL/04/1776	error - invalid character
10231990	error - incorrect date format
02/29/1982	error - invalid date

Coding CHARACTER values When loading CHARACTER values, X(4) is the minimum picture that will accommodate values exceeding the picture. That is, an item with a picture of X allows only one-character values; a picture of XX allows only one or two characters for each value; a picture of X(3) allows one, two, or three characters per value; and a picture of X(4) or larger allows values of up to 250 characters.

For CHARACTER values, trailing, leading, and extraneous (more than one embedded) blanks are ignored.

Loading values for a CHARACTER item with the specified pictures gives the following results:

Value Loaded for Item C1	Picture	Value Displayed with PRINT C1
1*A*	X(2)	1* A
1*ABC*	X(3)	1* ABC
1*AbC*	X(3)	1* AbC
1*ABCD*	X(3)	error-exceeds picture
1*ABCDE*	X(4)	1* ABCDE
1*bbABCDbbbEFGbbHIbbb	X(4)	1* ABCDbEFGbHI

Coding TEXT values TEXT and CHARACTER values differ only with regard to storing blanks. Values for TEXT items retain leading, trailing, and embedded blanks entered between the system separators.

Loading values for a TEXT item with the specified pictures gives the following results:

Value Loaded for Item C1	Picture	Value Displayed with PRINT C1
1*AbbBC*	X(4)	1*AbbBC
1*Abb*	X(2)	error - exceeds picture size
1*bAb*	X(2)	error - exceeds picture size
1*bbbABDbbbEFGbbb*	X(4)	1*bbbABDbbbEFGbbb

Coding UNDEFINED values The picture for an UNDEFINED item is in the form of X(n), similar to CHARACTER and TEXT item pictures. The default picture is X(4), that is, one fullword (four bytes). The minimum length is one byte, and the maximum is 250 bytes.

Your input format for UNDEFINED values can be either display or hexadecimal (IBM storage converted to display). The length of an UNDEFINED value must be equal to or less than the item's picture. The software will not truncate the value, and overflow is not allowed. Message -307- is issued if the value is longer than the picture.

If the data value for an UNDEFINED item is preceded by X and a single quote (X'), SYSTEM 2000 software expects valid hexadecimal data to follow. The parser will continue to accept data for the item until it encounters another single quote. The X-quote pair will be referred to here as *HEX notation.* Thus, valid HEX notation input has the following format:

X'*hexadecimal value*'

A hexadecimal value consists of pairs of hexadecimal digits, which are the integers 0 through 9 and the characters A through F. When combined in a display format (for example, most terminal input), the input value X'E2E8E2E3C5D440F2F0F0F0' would be converted to "SYSTEM 2000," that is, E2=S, E8=Y, and so forth.

HEX notation allows you to enter any character in an UNDEFINED value. Remember that a value of all binary zeros is treated as a missing value (null). An error occurs only if you omit the closing quote or if you enter an invalid hexadecimal digit.

If an UNDEFINED value is not in HEX notation, SYSTEM 2000 software expects it to be display format. That is, each character of input represents a character to be stored. Also, for display format, SYSTEM 2000 software stores every space, including trailing spaces, as hexadecimal 40.

For either input format, the system separator and entry terminator word must be given when required.

HEX notation is also acceptable for values of any other item type, such as INTEGER or DECIMAL values.

Note: If your data happen to contain a command terminator (;) or the system separator (the asterisk by default), results will be unpredictable. Also, HEX notation could cause errors if the system separator is set to be the single quote.

Efficiency Considerations

Criteria for sequencing logical entries Although logical entries do not have to be arranged in any particular order to be loaded, you can gain advantages by sequencing the entries. If you identify one or more items that are most likely to be used for qualification criteria for retrievals, then sequencing the data on those items may improve retrieval time.

Consider the EMPLOYEE database. Suppose that item C102 (DEPARTMENT) is used most frequently for retrievals and that item C7 (GENDER) is second in frequency. In that case, the input data for the EMPLOYEE database should be sequenced by C102 first and then by C7. For example, all entries for employees in the Marketing Department could be entered first with all male employees in that department placed before all female employees in the department.

Loading null records Sometimes, when a database is initially loaded, some or all of the entries are not available and must be entered at a later time. When the missing data cause entire data records to be omitted, inserting these data records later can cause data for an entry to be scattered among several pages of the Data Table. This data dispersal can increase the number of I/Os necessary to access data for those entries.

One way to remedy this situation is to enter null data records when the logical entry is initially loaded. For example, in the EMPLOYEE database, null C120 records (MONTHLY PAYROLL ACCOUNTING) can be entered for each employee. Then, as data become available each month, they can be added to these records.

To load null records, code only the record number followed by an asterisk. For example, the following loader stream loads twelve null C120 records for an employee:

```
1*1014*BROWN*...*100*101*...*120*120*120*120*120*120*120*120*120*120*
120*120*...*END*
```

Subsequently, a command like the following one adds values to the rightmost (*0) C120 data record:

```
ADD C120 *0 EQ 121*01/31/92*122*184.0*...*END* WHERE C1 EQ 1014:
```

If you use this loading method, take care when retrieving data if the NULL format option is in effect. (For details on the NULL format option, see Chapter 15, "Changing Output Format: Format Options.")

TYPICAL PROBLEMS WITH LOADING

This section presents some errors that might occur when you are preparing a loader stream. A careful check for these errors in a loader stream can make the loading process less time-consuming. The software will reject an entire entry if it finds an error in the loader stream for that entry.

To present these errors, a loader stream was prepared for seventeen entries. A session was run to load the loader stream into the EMPLOYEE database. The loader stream was placed in a data set with the DDname of LOADER.

The loader stream is shown below with each line identified by a number. Illus. 8.4 on page 8-22 shows the result of the software attempting to load the data. Notice that in the output, message -260- identifies each line containing an error as well as the entry where the error(s) occurs. Illus. 8.5 on page 8-24 shows the loader stream after corrections were made.

Illus. 8.3 Loader Stream with Errors

```
 1  1*1007*2*BROWN*3*VIRGINIA$5*05/24/1958*END*
 2  1*1067*2*CAHILL*3*JACOB*5*12/25/1950*END*
 3  1*1071*2*CANADY*3*FRANK*5*11/19/1953*13*1006*END*
 4  1*1077*2*GIBSON*100*103*GVHT*END*
 5  1*1031*2*CHAN*3*TAI*5*07/04/1958*13*1.6*END*
 6  1*1274*2*GARCIA*3*FRANCISCO*505/05/1967*END*
 7  1*1006*2*GARRETT*3*OLAN*5*01/23/47*310*END*
 8  1*1213*2*JOHNSON*3*BRADFORD*13*END*
 9  1*1145*2*JUAREZ*103*VPB*END*
10  1*1005*2*KNAPP*200*201*CPA*201*WRITING*END*
11  1*1024*2*MUELLER*100*PRK*END*
12  1*1112*2*JONES*110*114*176.20*END*
13  1*1004*2*KNIGHT*200*201*TYPING*110*112*MONTHLY*END*
14  1*1226*2*REDFOX*100*103*JC*110*112*MONTHLY*200*201*FORTRAN*
15  1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
16  100*103*CPW*105*01/02/89*END*
17  1*1219*2*KAATZ*3*FREDDIE*5**END*
18  1*1083*2*SAVAGE*3*WILLIAM*5*01/20/65*100*100*PQ*END*
19  *
```

The following typical errors are shown in Illus. 8.3 on page 8-19. To see the software's responses to each error, see Illus. 8.4 on page 8-22.

1. **Typing incorrect characters.**
   On line 1, VIRGINIA is followed by $ and not by an asterisk. The software considers VIRGINIA$5 as the value specified for item 3 and, expecting a component number next, scans 05/24/1958 as a component number. Since it is not, an error message appears.

   On line 2, the letter O instead of the digit 0 appears in the date.

   On line 18, the letter O instead of the digit 3 appears in the last 100* component number. Line 18 was not scanned because line 17 contains two consecutive asterisks. (See #8.)

2. **Exceeding picture for numeric data.**
   On line 3, 1006 is specified for item 13; however, the picture for item 13 is 999.

3. **Exceeding picture for CHARACTER or TEXT items with pictures of X, XX, or XXX.**
   On line 4, GVHT is specified for item 103; however, the picture for item 103 is XXX.

4. **Specifying incorrect values for numeric items.**
   On line 5, a decimal (1.6) is specified for item 13, which is defined as an INTEGER item.

5. **Omitting an asterisk after a component number.**
   On line 6, an asterisk is missing after component number 5. The software considers 505/05/1967 to be a component number, which it cannot recognize.

6. **Omitting an asterisk after a value.**
   On line 15, 310 (the value for item 13) is not followed by an asterisk. The software mistakes the 100 at the beginning of line 16 as a continuation of the value for item 13. Thus, the software issues the message that the value exceeds its picture, and, because it does not recognize the 100 as record number, it cannot recognize that 103 and 105 are items in record 100.

7. **Omitting an item number.**
   On line 7, 310 is specified as a value, but it is not preceded by its corresponding component number, which is 13. The software considers 310 to be a component number that it cannot recognize.

   On line 11, PRK is specified as a value, but it is not preceded by its corresponding component number, which is 103. After 100, the software expected a component number or END*.

8. **Specifying an item number and omitting a value for it.**
   On line 8, 13 is followed by END instead of a value.

   On line 17, 5* is followed by another asterisk instead of a value. The software interprets the two asterisks as an end-of-data marker and does not process line 18.

9. **Omitting record numbers before lower-level data records.**
   On line 9, 103 appears after a value for item 2 in C0. (A 100* is required before 103.)

On line 10, the second 201 appears after a value for item 201 in another data record. (A 200* is required before 201.)

On line 12, 110 appears after a value for item 2 in record C0. (A 100* is required before 110.)

On line 13, 110 appears after a value for item 201 in C200. (A 100* is required before 110.)

10. **Omitting END* at the end of each entry.**
In line 14, FORTRAN is not followed by END*. The software considers that the values in line 15 belong in entry 14. However, because the items in line 15 belong to a C0 record and not a C200 record, the software issues error messages for each item in line 15.

Notice that if line 16 had followed line 14, data in line 16 would become part of entry 14. Notice also that line 14 does not contain any errors. END* is missing at the end, but that does not cause an error until the software reads line 15.

Illus. 8.4 Sample Load of File LOADER

```
---
STOP AFTER SCAN IF ANY COMMAND IS REJECTED:
---
DATA FILE IS LOADER:
---
LOAD:

  -241- 13:13:24 - BEGIN LOADING -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       1, ENTRY NO.       1 -
1*1007*2*BROWN*3*VIRGINIA$5*05/24/1958*END*
.....................................D
  -809- SYNTAX ERROR IN COMPONENT NUMBER -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       2, ENTRY NO.       2 -
1*1067*2*CAHILL*3*JACOB*5*12/25/1950*END*
.....................................D
  -308- ERROR IN DATE OR NUMERIC VALUE -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       3, ENTRY NO.       3 -
1*1071*2*CANADY*3*FRANK*5*11/19/1953*13*1006*END*
.............................................D
  -307- VALUE EXCEEDS PICTURE WIDTH -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       4, ENTRY NO.       4 -
1*1077*2*GIBSON*100*103*GVHT*END*
............................D
  -307- VALUE EXCEEDS PICTURE WIDTH -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       5, ENTRY NO.       5 -
1*1031*2*CHAN*3*TAI*5*07/04/1958*13*1.6*END*
.....................................D
  -308- ERROR IN DATE OR NUMERIC VALUE -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       6, ENTRY NO.       6 -
1*1274*2*GARCIA*3*FRANCISCO*505/05/1967*END*
.....................................D
  -809- SYNTAX ERROR IN COMPONENT NUMBER -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       7, ENTRY NO.       7 -
1*1006*2*GARRETT*3*OLAN*5*01/23/47*310*END*
  -808- UNRECOGNIZED COMPONENT LABEL -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       8, ENTRY NO.       8 -
1*1213*2*JOHNSON*3*BRADFORD*13*END*
.................................D
  -308- ERROR IN DATE OR NUMERIC VALUE -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       8, ENTRY NO.       8 -
1*1213*2*JOHNSON*3*BRADFORD*13*END*
.................................D
  -210- UNEXPECTED END OF VALUE STREAM -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.       9, ENTRY NO.       9 -
1*1145*2*JUAREZ*103*VPB*END*
..................D
  -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.      10, ENTRY NO.      10 -
1*1005*2*KNAPP*200*201*CPA*201*WRITING*END*
.............................D
  -227- ITEM VALUED TWICE IN ONE RECORD -
  -260- ERROR IN DATA FILE LOADER   , CARD NO.      11, ENTRY NO.      11 -
1*1024*2*MUELLER*100*PRK*END*
.......................D
  -809- SYNTAX ERROR IN COMPONENT NUMBER -
```

Sample Load of File LOADER (continued)

```
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      12, ENTRY NO.       12 -
1*1112*2*JONES*110*114*176.20*END*
.................D
 -204- PARENT DATA RECORD IS MISSING -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      12, ENTRY NO.       12 -
1*1112*2*JONES*110*114*176.20*END*
.....................D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      13, ENTRY NO.       13 -
1*1004*2*KNIGHT*200*201*TYPING*110*112*MONTHLY*END*
..............................D
 -204- PARENT DATA RECORD IS MISSING -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      13, ENTRY NO.       13 -
1*1004*2*KNIGHT*200*201*TYPING*110*112*MONTHLY*END*
....................................D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      15, ENTRY NO.       14 -
1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
.D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      15, ENTRY NO.       14 -
1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
........D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      15, ENTRY NO.       14 -
1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
.................D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      15, ENTRY NO.       14 -
1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
............................D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      15, ENTRY NO.       14 -
1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310
.....................................D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      16, ENTRY NO.       14 -
100*103*CPW*105*01/02/89*END*
...D
 -307- VALUE EXCEEDS PICTURE WIDTH -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      16, ENTRY NO.       14 -
100*103*CPW*105*01/02/89*END*
.......D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      16, ENTRY NO.       14 -
100*103*CPW*105*01/02/89*END*
..............D
 -205- ITEM DOES NOT BELONG IN CURRENT RECORD -
 -260- ERROR IN DATA FILE LOADER   , CARD NO.      17, ENTRY NO.       15 -
1*1219*2*KAATZ*3*FREDDIE*5**END*
.........................D
 -209- UNEXPECTED END-OF-DATA ON DATA FILE -
 -240- 13:13:27 - SCAN COMPLETE -
 -226- ERROR LIMIT EXCEEDED -
```

Illus. 8.5 Corrected Loader Stream for LOADER

```
 1  1*1007*2*BROWN*3*VIRGINIA*5*05/24/1958*END*
 2  1*1067*2*CAHILL*3*JACOB*5*12/25/1950*END*
 3  1*1071*2*CANADY*3*FRANK*5*11/19/1953*13*106*END*
 4  1*1077*2*GIBSON*100*103*GVH*END*
 5  1*1031*2*CHAN*3*TAI*5*07/04/1958*13*106*END*
 6  1*1274*2*GARCIA*3*FRANCISCO*5*05/05/1967*END*
 7  1*1006*2*GARRETT*3*OLAN*5*01/23/47*13*310*END*
 8  1*1213*2*JOHNSON*3*BRADFORD*END*
 9  1*1145*2*JUAREZ*100*103*VPB*END*
10  1*1005*2*KNAPP*200*201*CPA*200*201*WRITING*END*
11  1*1024*2*MUELLER*100*103*PRK*END*
12  1*1112*2*JONES*100*110*114*176.20*END*
13  1*1004*2*KNIGHT*200*201*TYPING*100*110*112*MONTHLY*END*
14  1*1226*2*REDFOX*100*103*JC*110*112*MONTHLY*200*201*FORTRAN*END*
15  1*1003*2*SALAZAR*3*YOLANDA*5*12/12/1952*13*310*
16  100*103*CPW*105*01/02/89*END*
17  1*1219*2*KAATZ*3*FREDDIE*END*
18  1*1083*2*SAVAGE*3*WILLIAM*5*01/20/65*100*103*PQ*END*
19  *
```

THE LOADING PROCESS

This section provides a general description of how the LOAD command is processed. A knowledge of this process will help you to use the command more effectively.

It is important that you provide enough scratch file space before loading a database. Before you load any data, review the discussion of scratch file space in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*. The loading process consists basically of two parts: scanning the data to be loaded and updating the appropriate database tables.

Data Scanning

When you issue the LOAD command, SYSTEM 2000 software creates and processes APPEND TREE commands. As each APPEND TREE command is processed, SYSTEM 2000 software reads the value stream from the Data File until it finds an entry terminator (END*). If there are no errors, it places the value stream in Scratch File 7. Otherwise, it issues an informative message identifying the logical entry and the line being scanned. It also issues one or more messages indicating the errors in the value stream.

The system then proceeds to the next APPEND TREE command unless it finds two consecutive asterisks in the Data File. In that case, the scan stops, and SYSTEM 2000 software issues an informative message giving the time when the scan was completed.

When all the entries have been scanned, the system proceeds to update the database tables. However, no updating takes place if you have issued the DISABLE EXECUTION command. Also, no updating occurs if errors have occurred and you have issued a STOP AFTER SCAN IF command.

Updating the Database Tables

After scanning is complete, Scratch File 7 contains the error-free value streams for the logical entries to be loaded. The system uses the data in that file to update the Hierarchical Table. If possible, it uses any reusable space in the table. Also, as it updates the Hierarchical Table, it places in Scratch File 6 the data necessary to update the Data Table. When the modifications to the Hierarchical Table are complete, the system issues an informative message.

If reusable space was used, the system proceeds to sort Scratch File 6 and issues informative messages at the beginning and at the end of the sort. Using the contents of Scratch File 6, SYSTEM 2000 software updates the Data Table. When finished with the modifications to that table, it issues a message. If no key values were loaded, then the loading process stops. Otherwise, the system places the data necessary to update the indexes in Scratch File 2.

If the indexes are to be modified, the system sorts Scratch File 2 and issues informative messages at the beginning and at the end of the sort. Using the contents of Scratch File 2, the two tables that make up the indexes are updated. Because of the sorting of Scratch File 2, SYSTEM 2000 software places values for a given key item in contiguous pages and/or blocks, thus minimizing the number of I/Os used in future updates and retrievals.

If the definition of a key item includes a padding specification, the last block for that item in the Multiple Occurrence Table is padded as specified. Also, if the master password holder or DBA password holder has specified VALUES PADDING, the last page of values for each item in the Distinct Values Table is also padded.

If values that exceeded their picture were entered for CHARACTER or TEXT type key items, the system must update the Extended Field Table. Also, it must update the Data Table to include pointers to the Extended Field Table for each occurrence of such values. To do so, it sorts these long values and issues informative messages at the beginning and at the end of the sort. When the updating of the Data Table is completed, the system issues an informative message.

When all the tables are updated (that is, when the loading process is complete), the system issues an informative message, and you are given the QUEST processor.

Retrieving Data

Introduction

Creating a Collect File

Retrieving in Columnar Format

Retrieving in Sequential Format

Retrieving Key Value Information

Retrieving Value Streams

Changing Output Format

Specifying Output Order

Displaying Values Together

Retrieval Examples by Task

Chapter 9

Introduction

THE RETRIEVAL-CLAUSE 9-2
Format 9-2

RETRIEVAL COMMAND CONSIDERATIONS 9-3

RETRIEVAL COMMAND PROCESSING 9-4
Processing Retrieval Commands without a Where-clause 9-4
Processing Retrieval Commands with a Where-Clause 9-4
Processing Retrieval Commands with an Ordering-Clause 9-6
Processing Retrieval Commands with a By-Clause 9-6

OPTIMIZATION CONSIDERATIONS FOR RETRIEVING 9-8

INSERTING COMMENTS 9-9
Using the TITLE Option 9-9
Using the ECHO Command 9-9
Using Delimiters 9-9

The following commands are retrieval commands:

- LIST
- PRINT
- UNLOAD
- COLLECT

Each retrieval command consists of a retrieval-clause and an optional where-clause and ordering-clause.

Before you use these chapters for reference, be sure you have read Chapter 3, "Retrieving Data."

THE RETRIEVAL-CLAUSE

The retrieval-clause specifies a list of one or more retrieval objects (units) to be retrieved by a LIST, PRINT, UNLOAD, or COLLECT command. A retrieval-clause can include schema records, schema items, stored strings, stored functions, ad hoc functions, system functions, by-clauses, and Collect File items -- separated by commas.

You can follow a retrieval-clause with a where-clause or with both an ordering-clause and a where-clause.

Format

units

unit

```
|item
|record
|* function *
|* function (parameters)
|* string *
|* string (parameters)
|(adhoc-function)
|system-function
|by-clause
|cf-item
```

item is a schema item name or component number.

record is a schema record name or component number. A record cannot be specified in the retrieval-clause of a LIST command unless it appears in a by-phrase or with the COUNT system function.

function is a stored function name or component number.

string is a stored string name or component number.

parameters are the values that replace the numbered parameters in the stored function or string.

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions" for details.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions" for details.

by-clause specifies that only values for certain items are to be displayed in logical order. See Chapter 17, "Displaying Values Together: By-Clause" for details.

cf-item is a Collect File item name. The item name must be followed by (CF) if it has the same name as a schema component.

RETRIEVAL COMMAND CONSIDERATIONS

The following considerations apply to all retrieval commands.

- You must have R-authority for retrieval objects specified in a retrieval-clause and an ordering-clause. If the command includes a where-clause, you must have W-authority for the specified components.

- If a retrieval command includes a where-clause and no by-phrases are specified in the retrieval-clause, all retrieval objects must be related to the qualified records. That is, if the retrieval-clause specifies items in disjoint records and a where-clause is included, the retrieval-clause must contain a by-clause.

- If a retrieval command includes a where-clause and one or more by-phrases are specified in the retrieval-clause, the retrieval objects preceding the first by-phrase, as well as all records specified in the by-phrases, must be in the path of the target record. Also, the records in the by-phrases must be related to the records qualified by the where-clause.

- A retrieval-clause can contain a mixture of schema components and Collect File items if there is a where-clause containing a Collect File condition. If there is no where-clause, the retrieval-clause must contain either all Collect File items or all schema components (not mixed).

- You can specify an ordering-clause only in commands with a where-clause. Also, you cannot use Collect File items in an ordering-clause. If a Collect File condition produces more than one list of selected records, each list is sorted separately.

- The retrieval-clause is executed once for each separate list of selected records if more than one list is produced by a Collect File condition. For each execution, the singular set of values in the Collect File record is used.

- If a retrieval command includes a where-clause, the limits specified with a LIMIT command can affect the amount of output from the command. Limits are applied to each separate list of selected records if more than one list is produced due to a Collect File condition. See **Controlling Output Volume: LIMIT/END LIMIT** on page 31-2 for details on the LIMIT command.

- Format options specified in previous retrieval commands or in a previous FORMOP command affect the output format or arrangement of data for subsequent retrieval commands.

RETRIEVAL COMMAND PROCESSING

SYSTEM 2000 software processes retrieval commands according to whether the command has a where-clause, whether it has an ordering-clause, and whether it has a by-clause.

Processing Retrieval Commands without a Where-clause

To process a retrieval command without a where-clause, the software inspects every entry in the database to see whether it contains data for the components specified in the retrieval-clause. As each data record is inspected, the entire retrieval-clause is scanned to determine whether any item can be processed in or from that data record.

The database is inspected down to the lowest level necessary to obtain data for retrieval. For example, if only data from level 0 are requested, then only level 0 data records are inspected, even if the database is more than one level deep. This is comparable to manual retrieval: if the data to be retrieved are in the first line of each form, there is no need to inspect the entire form.

Processing Retrieval Commands with a Where-Clause

To process a retrieval command with a where-clause, the software inspects only those data trees containing records qualified by the where-clause. The software then finds the lowest level record that is either explicitly or implicitly specified in the retrieval-clause. This record is called the *target record.*

A record is *explicitly* specified if either its name or component number appears in the retrieval-clause. A record is *implicitly* specified if either the name or component number of an item in the record appears in the retrieval-clause.

The target record must be related to the type of record(s) found in the list of qualified records. That is, the target record must be a descendant, an ancestor, or a record of the same type as the record in the list of qualified records. If this relationship between the target record and the qualified records does not exist, the software cannot process the command.

Also, the objects in the retrieval-clause must be in the path of the target record in order for the command to be processed. (Objects in a by-clause are an exception to this rule.)

Once the target record is determined, the software compares the level of the target record with the level of the qualified records. If the levels are different, the qualified records are normalized to the level of the target record. That is, if the target record is an ancestor of the qualified records, then each qualified record is replaced by the ancestor record that is of the same type as the target record. If the target record is a descendant of the qualified records, each qualified record is replaced by those descendant records that are the same type as the target record. Thus, the list of qualified records is replaced by a new list of records at a different level. These new records are called *selected* records. If the qualified records and the target record are the same type, the qualified records automatically become selected records.

Before processing continues, the selected records are arranged in numerical order by Hierarchical Table addresses if they are not already in that order. The selected records are then accessed one at a time. If the retrieval-clause contains objects at a higher level, the

software obtains the Hierarchical Table addresses of the records pertaining to those objects. The retrieval-clause is then scanned from left-to-right, and the data are retrieved in that order without regard to level. Therefore, it is the input order of the objects in the retrieval-clause that determines the order of output for a selected data record.

This process is comparable to a manual retrieval system. Suppose a clerk is given a list of forms from which certain values are to be retrieved. The clerk would most likely arrange the forms in some order and start by reviewing the first one. The values would be retrieved from that form, and the next form would be processed. If the required values are found in the first and second sections of the form, the clerk would start focusing on the specific sections of each form. That is, specific sections would become a target, a starting point for retrieval.

If a Collect File item is specified in a retrieval command with a where-clause, the where-clause must also contain the Collect File item. That is, there must be at least one condition that compares values in a database to values in the Collect File. The where-clause is executed once for every record in the Collect File. For example, if there are eight records in the Collect File, then eight lists of qualified records are produced; some of the lists may be empty if the comparison fails to find any qualified records.

From each list of qualified records, a list of selected records is produced. Then the retrieval-clause is processed by looking at the first Collect File record and the first list of selected records and displaying the requested items. Then the second Collect File record and the second list of selected records are processed, and so on.

Basically, to process a retrieval command with a where-clause, the software

1. evaluates the where-clause to obtain the qualified data records.

2. determines the target schema record from the retrieval-clause.

3. obtains the selected data records from the qualified data records. If the qualified data records are already at the level of the target record, the qualified data records become the selected data records. Otherwise, the qualified data records are normalized to the level of the target schema record.

   If the where-clause contains Collect File conditions, the software processes the where-clause once for each Collect File record. A list of selected records is produced (could be empty) for each Collect File record. Then, the retrieval-clause is processed for each list of selected records.

4. sorts the selected data records into Hierarchical Table address order if they are not already in that order. If an ordering-clause is present, the Hierarchical Table addresses of the selected data records are sorted in the order specified by the sort keys.

5. accesses the selected data records one at a time. The retrieval-clause is scanned from left-to-right, and the data are retrieved in that order. Ancestors and descendants of selected data records could be required for processing the retrieval-clause.

Processing Retrieval Commands with an Ordering-Clause

When the software processes a command with a where-clause, it arranges the list of selected records in numerical order by Hierarchical Table address before processing the retrieval-clause. If the command also includes an ordering-clause, the software rearranges the list of selected records according to the specifications given in the ordering-clause, then processes the retrieval-clause.

If a sort key in the ordering-clause is an item or a function, its values are used for sorting. On the other hand, if a sort key is a schema record, the Hierarchical Table addresses of the corresponding data records are used for sorting. Collect File items cannot be used as sort keys.

Processing Retrieval Commands with a By-Clause

You can use a by-clause to specify that only values for certain items within a tree are to be displayed together in logical order.

By-clause processing is identical in concept to the processing used with a PRINT *record* command. For example, the command PRINT CO causes the entire database, including values from disjoint records, to be printed in logical order. The only difference between by-clause processing and a PRINT *record* command is that with PRINT *record*, all values for items in data trees topped by records similar to the specified record are displayed in logical order, while by-clause processing results in the display of only those items specified in the by-clause.

When the software finds a by-clause, it focuses on the record specified in the by-phrase (the record specified after the word BY) and treats the whole by-clause as if the record alone were specified, except that it retrieves values only for the objects specified after the by-phrase.

To determine the target record in a retrieval-clause with a by-clause, the software selects one of the following:

- the lowest level component preceding the first by-phrase
- the lowest level record specified in a by-phrase.

If the command includes a where-clause and one or more by-clauses, the schema record(s) in the by-phrase(s) must be related to the records qualified by the where-clause, although the objects within a by-clause need not be related to each other or to the qualified records. Furthermore, the objects preceding the first by-phrase and the schema records in the by-phrases must be in the path of the target record.

Because of the manner in which the software processes by-clauses, there are three cases in which you can use by-clauses. These cases are presented below.

Case 1. You can use a by-clause to retrieve data from disjoint records when a where-clause is included in the command. When a where-clause is not included in a command, values from disjoint records can be retrieved because the software inspects the database in logical order.

Case 2. You can use a by-clause to retrieve data from disjoint records among those records qualified in the where-clause. For example, the following command cannot be processed because the target record (C110) and the qualified record (C300) are disjoint:

```
PR C113 WH C302 EQ AUSTIN CHESS CLUB:
```

There are two ways to retrieve data from one record and qualify from a disjoint record. One way is to use the HAS operator in the where-clause. The other way is to use a by-clause in the retrieval-clause. For example, the following command produces the output shown below it:

```
PR C113 WH C0 HAS C302 EQ AUSTIN CHESS CLUB:

113* 03/01/1990  ←  from record 14
113* 03/01/1989  ←  from record 15
113* 02/16/1988  ←  from record 17
113* 03/01/1991  ←  from record 38
```

The qualified record is a C0 record (the parent of the C300 record where C302 equals AUSTIN CHESS CLUB), the target record is C110, and the selected records are the two C110 records that are descendants of the qualified record. The output is displayed in physical order.

On the other hand, this command produces different output:

```
PR BY C0, C113 WH C302 EQ AUSTIN CHESS CLUB:

113* 03/01/1991  ←  from record 38
113* 03/01/1990  ←  from record 14
113* 03/01/1989  ←  from record 15
113* 02/16/1988  ←  from record 17
```

The qualified record is a C300 record, the target record is C0, and the selected record is the C0 record that is the parent of the qualified C300 record. The output is displayed in logical order.

Case 3. You can use a by-clause to retrieve data in logical order when a where-clause is included in a command.

OPTIMIZATION CONSIDERATIONS FOR RETRIEVING

Consider the following techniques for optimizing retrieval commands:

- Where applicable, use the DITTO operator to repeat an action-clause in consecutive commands.

- To obtain counts for key items, use the TALLY command rather than PRINT COUNT *item*. TALLY uses the index. PRINT COUNT scans the entire Data Table. However, the total number of distinct values from TALLY or PRINT COUNT does not include nulls.

  The most efficient way of including nulls in the count is to use PRINT COUNT *record*. This involves a scan of the Hierarchical Table only, which saves Data Table I/O. Notice that where-clauses should not be used for total counts. That is, a command such as the following one is not only costly, but also redundant:

  ```
  PRINT COUNT C1 WHERE C1 EXISTS:
  ```

- For quick reports, use LIST commands with default headings. The LIST output is more compact than the vertical output of the PRINT command.

INSERTING COMMENTS

You can use the following methods to insert comments between retrieval commands.

Using the TITLE Option

If the command ECHO ON is in effect, the echo of every command is sent to the Message File. Therefore, you can insert text following the keyword TITLE, and the text is displayed with the command in the Message File, which is usually the same as the Report File.

For example, if you issue the following command, it is displayed in the Message File, as shown, and is interleaved between the output on the Report File, provided both files are the same physical file:

```
PRINT /TITLE THIS TEXT IS DISPLAYED IN THE MESSAGE FILE/:
```

A comment can be any string of characters not containing the command terminator or a slash. The text cannot begin with a left parenthesis or with the letters B, D, F, L, R, or S followed by a left parenthesis. The text can be no more than 129 characters long; however, leading, trailing, and extraneous (more than one embedded) blanks are not counted, even though they are displayed. The total number of characters, including all blanks, cannot exceed 250.

The PRINT, UNLOAD, and COLLECT commands allow the TITLE option; however, the software does not process the option, and no action takes place except in the case of the COLLECT command.

If a COLLECT command contains just a TITLE comment (and no Collect File items are specified), the command cancels the Collect File as though the COLLECT command had been issued, and an error message is displayed. (The TITLE option is processed only within LIST commands.)

Using the ECHO Command

An alternate method of placing comments directly on the Report File is available through the use of the ECHO */string/* command. See Chapter 43, "Echoing Commands: ECHO" for information about the ECHO command.

Using Delimiters

You can also insert a comment after any CONTROL, DEFINE, QUEST, or REPORT language command by using the available comment syntax. You can insert a comment anywhere after a command terminator and before the next command. ECHO ON, however, must be in effect to write the comments to the Message File. See Chapter 44, "Inserting Comments" for details.

Chapter 10

Creating a Collect File: COLLECT

Formats 10-2
Collect File Item Names 10-4
Considerations 10-4
Collect File Item Picture 10-5
Processing a COLLECT Command 10-5
Processing without a where-clause 10-5
Processing with a where-clause 10-5
Writing a record to the Collect File 10-6
Using a Collect File 10-7
Example: Three Databases, One Collect File 10-7

The COLLECT command creates a relational table called the *Collect File*, which contains values extracted from one or more databases. In a COLLECT command retrieval-clause, you specify items from the current database and/or items from a previous Collect File that are to be used to build a new Collect File.

You can consider a Collect File a temporary database. SYSTEM 2000 software maintains a definition of the Collect File that you can display with the DESCRIBE command. To display Collect File values, use the PRINT, LIST, and UNLOAD commands or the REPORT language.

A Collect File remains intact until you issue another COLLECT command or exit the software. Therefore, if you switch databases, you can *carry* data or *link* records from one database to another.

Formats

```
COLLECT [/[options] [,TITLE comment]/] retrieval-clause
    [[,ordering-clause] where-clause] :
```

```
COLLECT :
```

options is a list of one or more of the following format options, separated by commas. Defaults are listed in the first column.

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

See Chapter 15, "Changing Output Format: Format Options" for details on format options.

TITLE including the TITLE option allows you to insert a comment within the output. Include a comma before the keyword TITLE only if you specify format options.

comment can be any string of characters not containing the command terminator or a slash, and not beginning with a left parenthesis or with the letters B, D, F, L, R, or S followed by a left parenthesis. The comment is displayed with the echo of the command in the Message File. See **Using the TITLE Option** on page 9-9 for more information.

retrieval-clause specifies a list of one or more Collect File (CF) retrieval objects, separated by commas. See **The Retrieval-Clause** on page 9-2 for details.

CF-retrieval-object [*cf-item* =] *retrieval-object*

cf-item is a unique Collect File item name, which can be any character string. If reserved words or component numbers are used, the Collect File cannot be used to create a new database. The Collect File item name can be omitted if it is the same as the retrieval object's name. Collect File items do not have any component numbers associated with them except in DESCRIBE and UNLOAD output.

A Collect File can have a maximum of 59 items.

Collect File items do not have key or non-key status. The software simply examines each Collect File item, record by record, when processing a Collect File. (However, the DESCRIBE output treats Collect File items as key items.)

retrieval-object is a schema item name or component number, a Collect File item named, stored function, stored string, system function, ad hoc function, or by-clause. You cannot specify a schema record unless it is embedded in a system function.

ordering-clause specifies the order of selected data records. See Chapter 16, "Specifying Output Order: Ordering-Clause" for details. If you specify an ordering-clause, you must also include a where-clause. You cannot specify Collect File items in an ordering-clause.

where-clause selects specific data records. See Chapter 24, "The Where-Clause" for details.

When using a Collect File item in a where-clause, the condition must be an item-to-item comparison with the Collect File item as the right-hand term.

The first format of the COLLECT command writes the values of the items specified in the retrieval-clause to a Collect File. The second format (the COLLECT verb only) cancels a Collect File.

A Collect File consists of variable length records, one record for each set of values. Each record contains the same number and type of items. The lengths of the records vary because of the different lengths of CHARACTER, TEXT, and UNDEFINED values. For each retrieval object in a COLLECT command retrieval-clause, there must be a corresponding Collect File item.

A Collect File requires the use of scratch pads. A scratch pad is a scratch file that is not accessible to you. (See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for information about scratch pads.) A Collect File can be described, displayed, and unloaded. Also, by using the /DEFINE/ option in the DESCRIBE command and the UNLOAD/COLLECT FILE/ command, you can create a new database from the Collect File.

If a COLLECT command contains both existing Collect File items and database items, a where-clause with a Collect File condition is required to associate the database values with the Collect File items.

You can also use Collect File items in subsequent COLLECT commands. Through the use of the where-clause and through switching databases, the COLLECT command links and gathers values from multiple databases. As part of building the Collect File, the software creates a definition for the Collect File to define the items in the file. This definition applies when the Collect File is used in subsequent commands. The definition contains the name, type, and picture for each Collect File item. The following topics describe how this information is obtained.

Collect File Item Names

You can name each Collect File item in the COLLECT command retrieval-clause. The default Collect File item name is the retrieval object name. You cannot use reserved words, such as WHERE, SUM, and COUNT, as the first word in a Collect File name. For example, suppose an ORDERS database having a schema item named ORDER DATE contains only one retrieval object, ORDER DATE. Therefore, there is only one Collect File item in each of the records in the Collect File. The name of this Collect File item is ORD DATE.

```
COLLECT ORD DAT = ORDER DATE WHERE ... :
```

If a retrieval object in the COLLECT command is identical to the Collect File item name, the Collect File item name can be omitted. For example, the command

```
COLLECT ORDER NUMBER, FUNNAME = COUNT PART NUMBER WHERE ... :
```

is equivalent to this command:

```
COLLECT ORDER NUMBER = ORDER NUMBER, FUNNAME = COUNT PART NUMBER WH ... :
```

In this example, there are two retrieval objects. Therefore, there are two Collect File items in each Collect File record. The first Collect File item name is ORDER NUMBER. The second retrieval object is COUNT PART NUMBER, which is a system function. The name FUNNAME is assigned to this second Collect File item.

If a Collect File item has the same name as a schema item or stored function and that name is used in a where-clause or a retrieval-clause, the software assumes that the schema item or stored function is being referenced. It does not recognize the reference as a Collect File item. To reference a Collect File item that has the same name as a schema item name, you must specify the (CF) notation after the item name. For example:

```
PRINT ORDER NUMBER WHERE ... :
```

prints order number values from the database. Whereas,

```
PRINT ORDER NUMBER (CF) WHERE ... :
```

prints order number values from the Collect File.

The name assigned to a Collect File item must be unique within the Collect File. If you attempt to specify the same name to more than one Collect File item in a COLLECT command, the software issues a message, ignores the command, and continues processing. The contents of the Collect File, if any, are undisturbed.

Considerations

- The NULL/NULL SUPPRESS format option has no effect on records written to a Collect File.
- The ZERO/ZERO SUPPRESS format option causes zeros to be used in functions instead of nulls.
- The REPEAT/REPEAT SUPPRESS format option affects records being written to the Collect File.

- If you specify an ad hoc function in a COLLECT command, the Collect File item type is determined by the type of the last item (or constant) in the function. For example, suppose C1 is a DATE item type, COLLECT (C1 + 20) produces an INTEGER Collect File item whose values equal the elapsed days in the C1 values plus 20. However, COLLECT (20 + C1) produces a DATE Collect File item whose values represent the C1 dates plus 20 days.

Collect File Item Picture

For each Collect File item, the Collect File definition contains a description of the characteristics of the item. Such characteristics include the item type and picture. The picture of the Collect File item depends solely upon the retrieval object from which it is built. If a Collect File item is built from either a schema item or a Collect File item, then the Collect File item's picture is the same as that of the schema item. If stored functions, ad hoc functions, or system functions are used for Collect File items, the picture is 9(10).9(3) unless the function is type INTEGER, which has a picture of 9(15).

CHARACTER, TEXT, and UNDEFINED items are stored in the Collect File as variable length fields with the complete value retained. The length of the field depends upon the length of the data values. The defined item length determines the default column width for a CHARACTER, TEXT, or UNDEFINED Collect File item used in a LIST command. For numeric pictures, the length of the field in the Collect File record is the number of words required to contain the value in the internal format, that is, in packed decimal format or floating point.

Processing a COLLECT Command

As with the other retrieval commands, the processing done on the retrieval-clause depends on whether the command contains a where-clause, on whether the command contains functions, and on the level at which the functions are evaluated. The possible combinations are discussed below.

Processing without a where-clause Similar to the PRINT, LIST, and UNLOAD commands, processing is done in database logical order. Evaluation of the retrieval objects occurs in the following order:

1. any functions that are to be evaluated either implicitly or explicitly over the entire database. (That is, the function is not contained in a by-phrase or is not contained in any by-clause, or, if it is contained within a by-clause, the by-clause is BY DATA BASE.)

2. any retrieval objects that are contained in an explicit BY DATA BASE by-clause.

3. any other retrieval objects, which are evaluated when the schema record in which they are contained is found during traversal of the database. If contained within a by-phrase or by-clause, the retrieval object is evaluated when an occurrence of the schema record specified in the by-phrase is found.

Processing with a where-clause After the list of selected records has been generated and after the ordering-clause (if any) has been processed, generation of Collect File records begins. As is the case for all retrieval-clauses acting on a list of selected records, the software processes the retrieval-clause in the following order:

1. From left-to-right in the retrieval-clause, the software evaluates

   a) any functions not contained within a by-phrase or not contained in a by-clause (with the exception of a BY DATA BASE by-clause). The domain for the function evaluation is the list of selected records.

   b) any retrieval objects contained within a BY DATA BASE by-clause. The domain for their evaluation is the entire database; the list of selected records is ignored. Functions within the BY DATA BASE by-clause are evaluated before any other retrieval objects in the by-clause.

2. The software reads a pointer from the list of selected records. The list of selected records has been normalized to the level of the deepest retrieval object in the retrieval-clause. The path is determined between the data record associated with the pointer and the data record at the level of the highest level item in any retrieval object. Then, the retrieval-clause is processed from left-to-right, evaluating any retrieval object found. Retrieval objects evaluated previously are not reevaluated.

If the number of selected records is zero, then the retrieval-clause of the COLLECT command is executed to process only Collect File items that are retrieval objects and that are not associated with a schema item, that is, those that are not within a function such as SUM (schema item + CF item). If there are no Collect File items to be processed, then the Collect File contains no data, and the Collect File definition is discarded.

Writing a record to the Collect File A record is written to the Collect File by following the path of each selected record. If, for a given path, the software is unable to descend to the deepest item, those items that are missing are considered to be null, and a null is written in that item's field in the Collect File record. If an item value is null in a data record, then a null is written to the Collect File.

The following example shows processing a null item using the PARTS database (see Illus. 10.1 on page 10-8 and Illus. 10.3 on page 10-10).

```
COLLECT PART NUMBER, VENDOR NUMBER:
```

The Collect File is built with the following data:

	PART NUMBER	VENDOR NUMBER
record 1	101	1299
record 2	101	1323
record 3	201	-NULL-
record 4	522	1592
record 5	522	1680
record 6	629	1402
record 7	629	1892

Note that in record 3, there is no VENDOR NUMBER for PART NUMBER 201. However, the Collect File record is written with the VENDOR NUMBER field containing a null.

For a COLLECT command without a where-clause, there is at least one Collect File record written for each record occurrence of the highest level item in the retrieval-clause that is a retrieval object. If that item happens to be null in a given record occurrence, a Collect File record is still written, and the value for that item in the Collect File is null.

For a COLLECT command with a where-clause, there is one Collect File record for each selected record. Due to the inclusive normalization done when the list of qualified records is normalized to the level of the target record type, the list of selected records contains the pointer to the deepest record in each path that must be written to the Collect File.

If one or more retrieval objects are within a by-clause, then when a Collect File record is written due to processing a path within that by-clause, all retrieval objects outside that by-clause will have null values on the record.

The REPEAT SUPPRESS format option has the following effect on the records written to the Collect File:

- When a schema item's value is written to the Collect File, null is written if the item's value was written previously. (That is, schema items behave the same way with respect to REPEAT SUPPRESS as they do in the other retrieval commands.)

- When a Collect File item's value is written to the Collect File, null is written if the value is the same as that written to the previous Collect File record.

Using a Collect File

You can use the data collected in a Collect File in other commands, including another COLLECT command. The items in the Collect File can be used in a QUEST command retrieval-clause or where-clause. You cannot, however, use Collect File items in an update command or in CONTROL or DEFINE language commands.

If, in parsing a QUEST command, the software finds an item that is not part of the current database's schema or that is specified as a Collect File item by (CF) after the name, the Collect File is inspected to see whether its definition contains the item. If so, the software assumes that the Collect File will be used in processing the retrieval command. If the Collect File definition does not contain the item, an error message is issued.

A Collect File item can be used just like a schema item in a retrieval-clause. In a where-clause, a Collect File item can appear only as the right-hand term of an item-to-item comparison condition, called a Collect File condition.

For more information about processing a where-clause that has one or more Collect File conditions, see Chapter 24, "The Where-Clause."

Example: Three Databases, One Collect File

You can use a Collect File to pass data from one database to another.

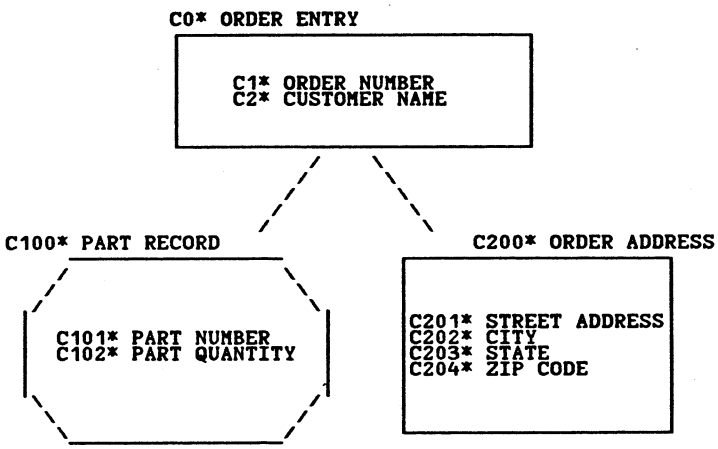
Suppose you want to list the current orders, along with the name of the customer placing the order, the name of the part that you ordered, and the vendor's name of the part being

ordered. This query requires data from three databases: the ORDERS database for the order numbers and customer names, the PARTS database for the part names, and the VENDORS database for the vendor names.

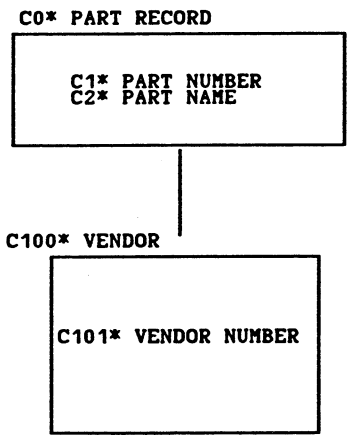
The following illustrations show the schemas for the three databases. Illus. 10.2 on page 10-9, Illus. 10.3 on page 10-10, and Illus. 10.4 on page 10-11 show the data records for the ORDERS, PARTS, and VENDORS databases, respectively.

Illus. 10.1 Schemas for the ORDERS, PARTS, and VENDORS Databases

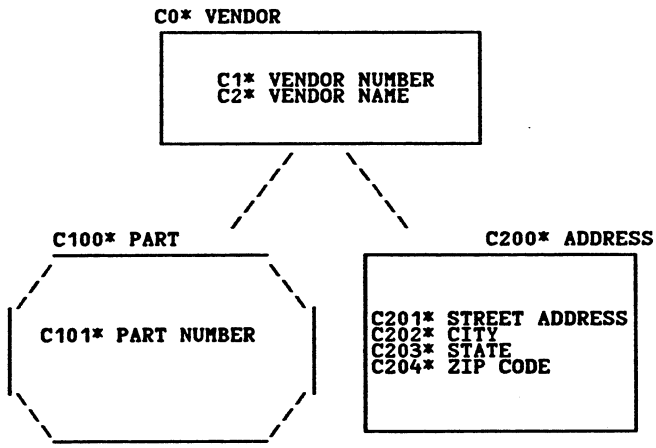
ORDERS Database Schema



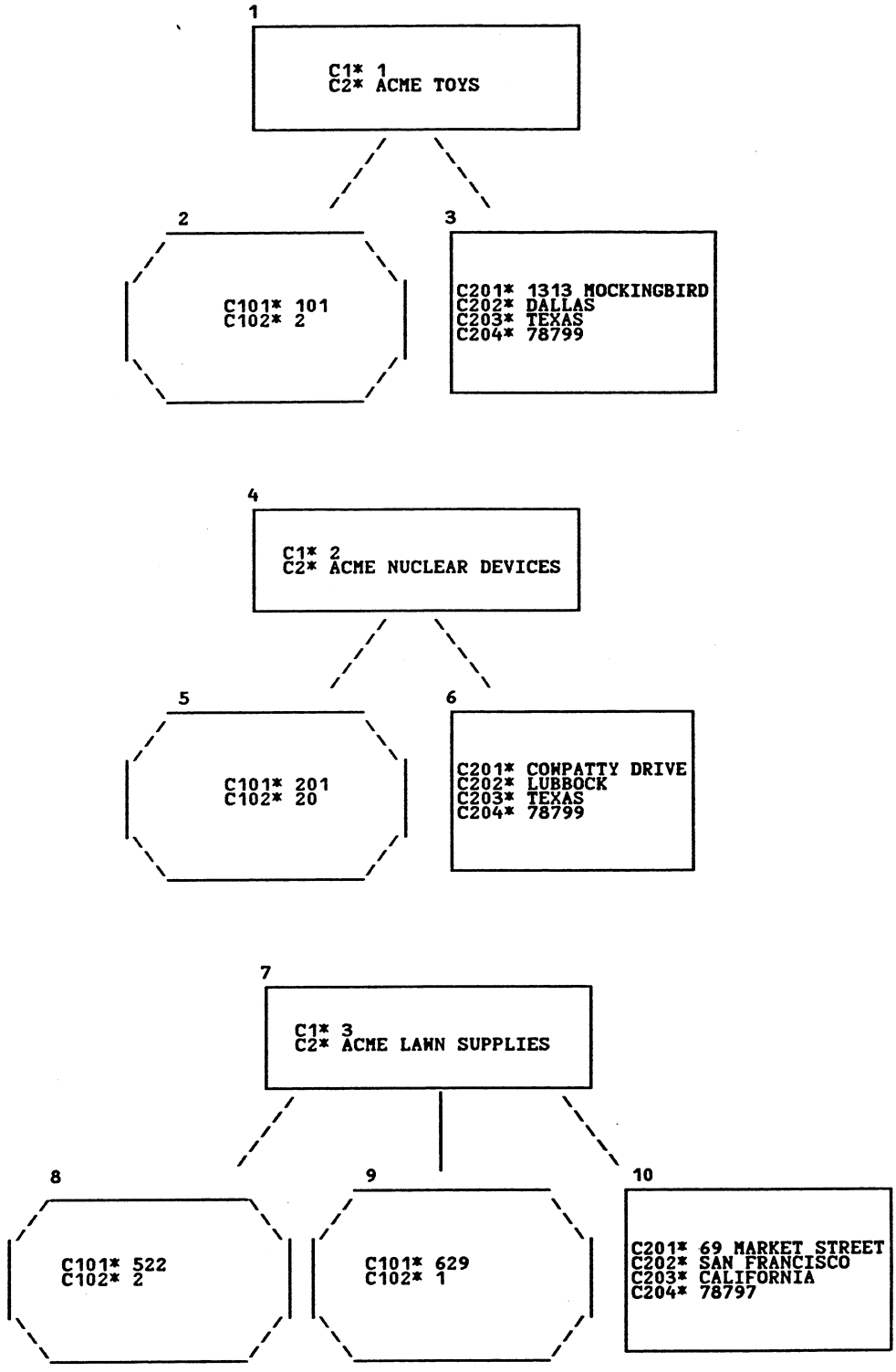
PARTS Database Schema



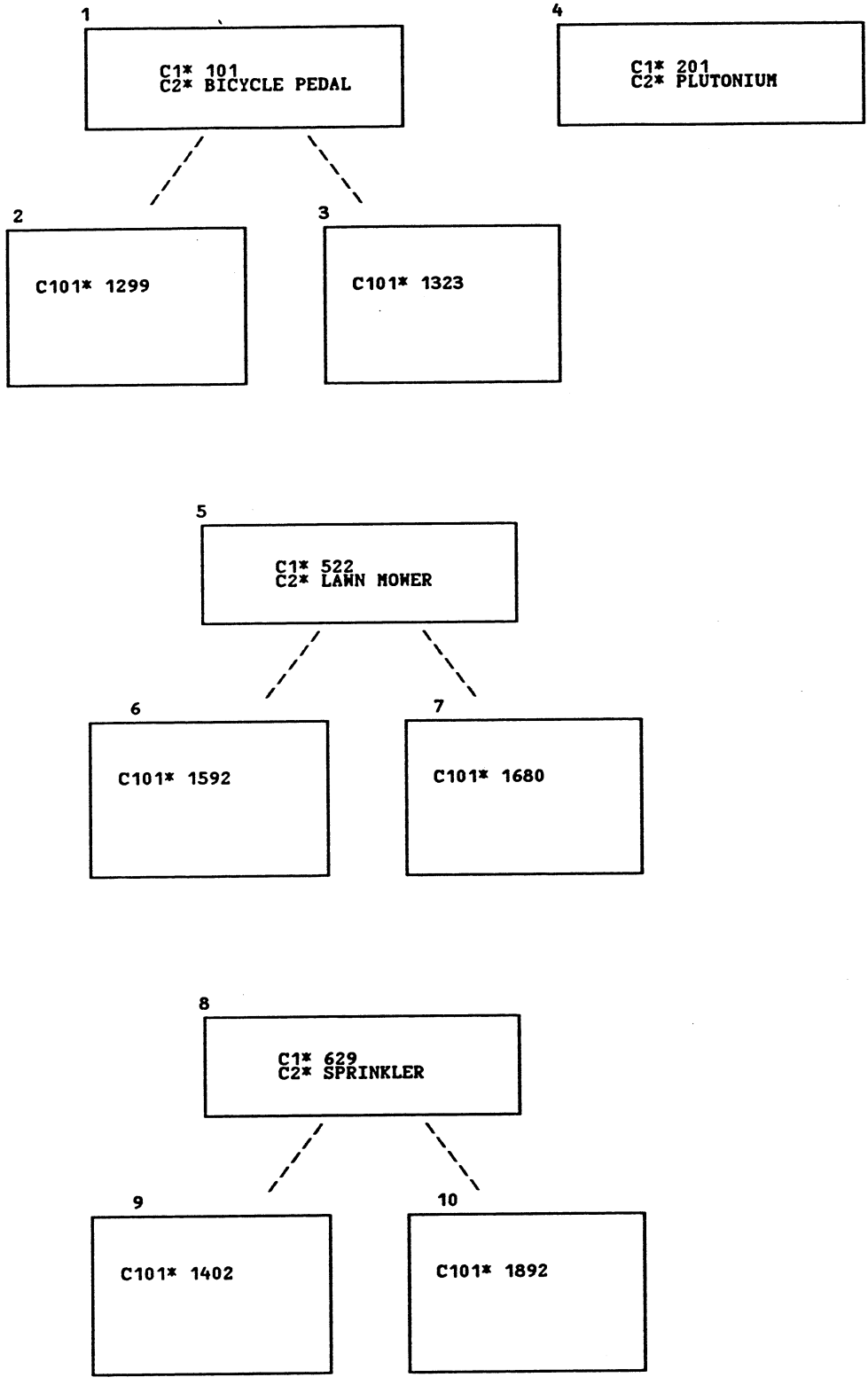
VENDORS Database Schema



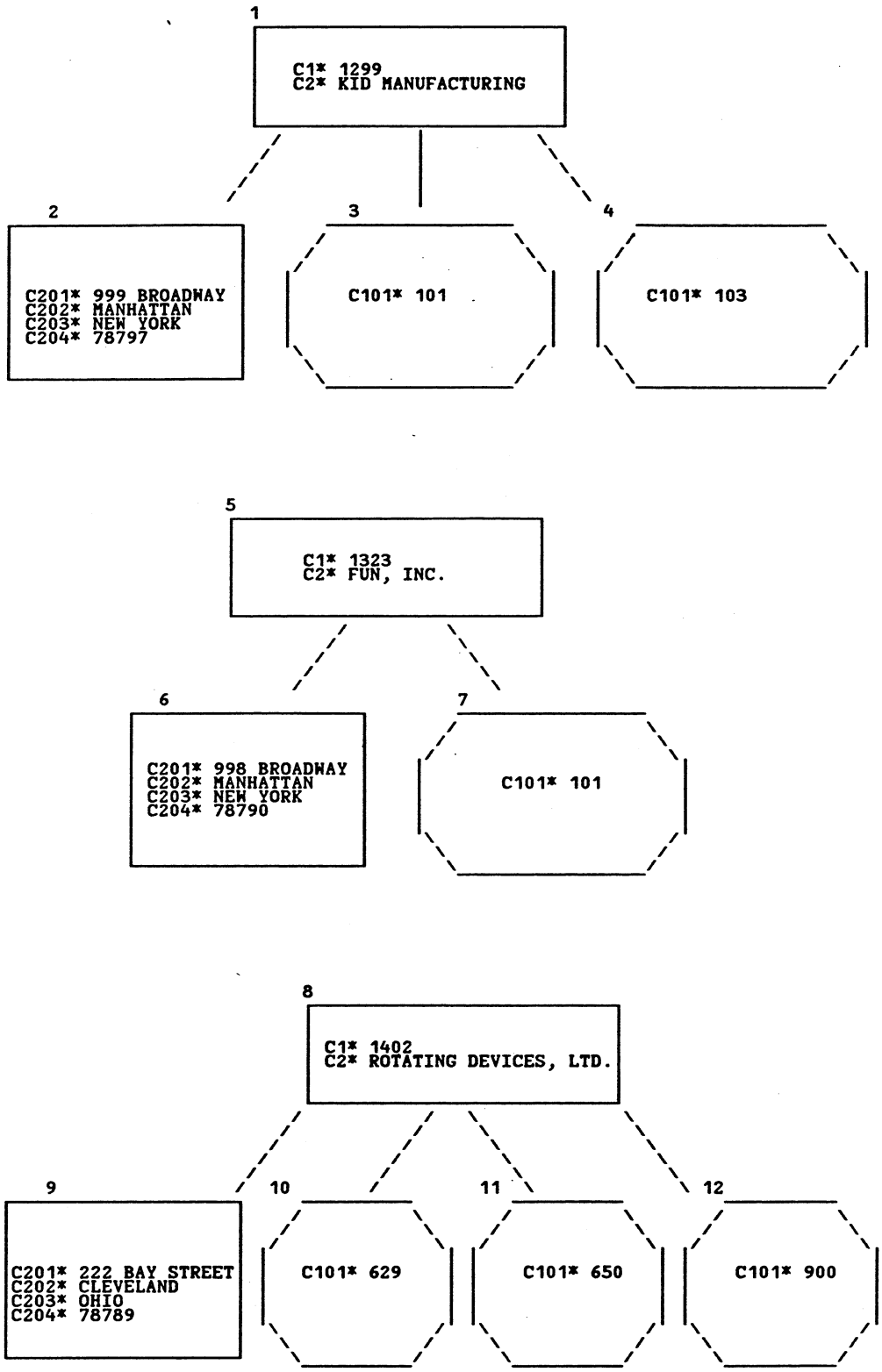
Illus. 10.2 Records in ORDERS Database

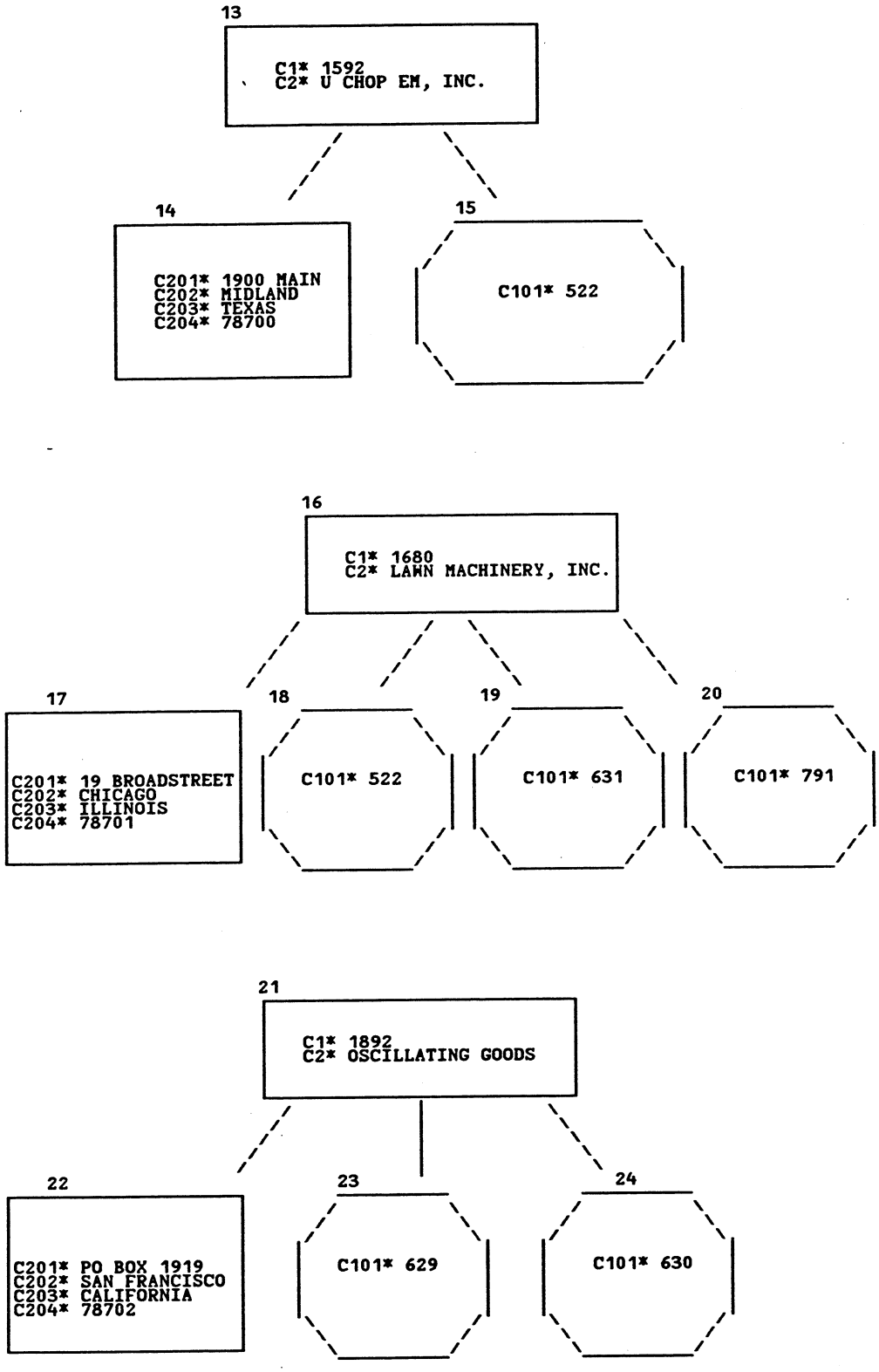


Illus. 10.3 Records in PARTS Database



Illus. 10.4 Records in VENDORS Database




13
C1* 1592
C2* U CHOP EM, INC.
14
C201* 1900 MAIN
C202* MIDLAND
C203* TEXAS
C204* 78700
15
C101* 522
16
C1* 1680
C2* LAWN MACHINERY, INC.
17
C201* 19 BROADSTREET
C202* CHICAGO
C203* ILLINOIS
C204* 78701
18
C101* 522
19
C101* 631
20
C101* 791
21
C1* 1892
C2* OSCILLATING GOODS
22
C201* PO BOX 1919
C202* SAN FRANCISCO
C203* CALIFORNIA
C204* 78702
23
C101* 629
24
C101* 630

To begin, the ORDERS database must be accessed to obtain the order numbers and the customers' names. To get the parts' names, attach the PARTS database. The ORDERS and PARTS databases are related to one another via PART NUMBER. The first step is to build a Collect File containing order numbers, customer names, and the part numbers for a given order.

The following commands create the Collect File shown below:

```
USER, X:
DBN IS ORDERS:
COLLECT ORDER NUMBER, CUSTOMER NAME, PART NUM = PART NUMBER
   WHERE C1 EXISTS:
```

	ORDER NUMBER	CUSTOMER NAME	PART NUM
record 1	1	ACME TOYS	101
record 2	2	ACME NUCLEAR DEVICES	201
record 3	3	ACME LAWN SUPPLIES	522
record 4	3	ACME LAWN SUPPLIES	629

Next, the PARTS database must be accessed to obtain the part names. As previously mentioned, the PARTS and VENDORS databases are related to each other via the vendor number field. So, the vendor number field is obtained from the PARTS database.

First, the PARTS database must become the current database; therefore, the following commands are issued to switch to the PARTS database. (You do not exit the software to do this.)

```
CONTROL:
USER, ABC:
DBN IS PARTS:
```

Using the Collect File as input, another COLLECT command is then issued to rebuild the Collect File.

```
COLLECT ORDER NUMBER, CUSTOMER NAME, PART NAME,
   VEN NUMBER = VENDOR NUMBER WH PART NUMBER* EQ PART NUM*:
```

In the retrieval-clause of this COLLECT command, there are four retrieval objects: ORDER NUMBER, CUSTOMER NAME, PART NAME, and VEN NUMBER (which is VENDOR NUMBER in the PARTS database). The software recognizes that ORDER NUMBER and CUSTOMER NAME are items in the Collect File because these items are not in the PARTS database definition and are in the Collect File. Likewise, in the where-clause, the software recognizes that PART NUM is a Collect File item because it is not an item in the PARTS database.

Note that in the first COLLECT command issued on the ORDERS database, the part number field was assigned the name of PART NUM in the Collect File. If this had not been done, the Collect File item would have been assigned the name PART NUMBER by default. Because the PARTS database contains a schema item named PART NUMBER, you would have had to specify PART NUMBER (CF) for the software to recognize PART NUMBER as a Collect File item.

Because the where-clause contains a Collect File item, it is processed once for each Collect File record. In this example, there are four Collect File records, so the where-clause is processed four times. The first execution of the where-clause is processed as

```
PART NUMBER EQ 101
```

This condition qualifies record 1 in the PARTS database. The selected records are 2 and 3.

The second execution of the where-clause is processed as

```
PART NUMBER EQ 201
```

This condition qualifies record 4 in the PARTS database. The selected record is also record 4 because there are no level 1 records and normalization is inclusive.

The third execution of the where-clause is processed as

```
PART NUMBER EQ 522
```

This condition qualifies record 5 in the PARTS database. The selected records are 6 and 7.

The fourth and final execution of the where-clause is processed as

```
PART NUMBER EQ 629
```

This condition qualifies record 8 in the PARTS database. The selected records are 9 and 10.

Therefore, at the end of the where-clause processing, there are four lists of selected records: one list for each Collect File record. The following chart summarizes the processing so far:

INTERMEDIATE FILE SELECTED RECORD LIST

record 1	record 2	record 3	record 4
2	4	6	9
3		7	10

Now, the retrieval-clause is processed. Because the retrieval-clause contains Collect File items, it is processed once for each Collect File record, using the list of selected records associated with that Collect File record.

Therefore, in processing the first Collect File record, the following Collect File records are written:

	ORDER NUMBER	CUSTOMER NAME	PART NAME	VEN NUMBER
rec 1	1	ACME TOYS	BICYCLE PEDAL	1299
rec 2	1	ACME TOYS	BICYCLE PEDAL	1323

Notice that the values for ORDER NUMBER and CUSTOMER NAME are obtained from the Collect File record.

After the retrieval-clause is processed for the first Collect File record, the second Collect File record and its list of selected records is used.

	ORDER NUMBER	CUSTOMER NAME	PART NAME	VEN NUMBER
rec 3	2	ACME NUCLEAR DEVICES	PLUTONIUM	-NULL-

Note that, in this case, there is no VENDOR NUMBER for this entry; therefore, a null is written to the Collect File.

Continuing with the third Collect File record and its selected record list:

	ORDER NUMBER	CUSTOMER NAME	PART NAME	VEN NUMBER
rec 4	3	ACME LAWN SUPPLIES	LAWN MOWER	1592
rec 5	3	ACME LAWN SUPPLIES	LAWN MOWER	1680

The fourth and last Collect File record and selected record list is as follows:

	ORDER NUMBER	CUSTOMER NAME	PART NAME	VEN NUMBER
rec 6	4	ACME LAWN SUPPLIES	SPRINKLER	1402
rec 7	4	ACME LAWN SUPPLIES	SPRINKLER	1892

At this point, the only item still to be obtained is the vendor name. Therefore, the VENDORS database must be accessed, using the following commands:

```
CONTROL:
USER, XYZ:
DBN IS VENDORS:
```

There is no need to build another Collect File because the data currently in the Collect File plus the data from the VENDORS database are all the data required to produce the listing needed. Therefore, the following LIST command can be issued:

```
LIST ORDER NUMBER, CUSTOMER NAME, PART NAME, VENDOR NAME
   WHERE VENDOR NUMBER* EQ VEN NUMBER*:
```

In this command, ORDER NUMBER, CUSTOMER NAME, PART NAME, and VEN NUMBER are Collect File items. (Notice that in the second COLLECT command, VEN NUMBER was assigned to be the Collect File item name to avoid confusion with the schema item.) Because there is a Collect File item in the where-clause, the where-clause is processed once for each Collect File record. For the first Collect File record, the following condition is processed:

```
VENDOR NUMBER EQ 1299
```

This results in record 1 of the VENDORS database being qualified and selected (no normalization is needed).

The condition for the second Collect File record is

```
VENDOR NUMBER EQ 1323
```

This results in record 5 of the VENDORS database being qualified and selected.

The condition for the third Collect File record is

```
VENDOR NUMBER EQ -NULL-
```

This results in a null list since a null Collect File value stops that execution of the where-clause.

For the fourth Collect File record, the condition is

```
VENDOR NUMBER EQ 1592
```

This results in record 13 of the VENDORS database being qualified and selected.

The condition for the fifth Collect File record is

```
VENDOR NUMBER EQ 1680
```

This results in record 16 of the VENDORS database being qualified and selected.

The condition for the sixth Collect File record is

```
VENDOR NUMBER EQ 1402
```

This results in record 8 of the VENDORS database being qualified and selected.

For the seventh and last Collect File record, the condition is

```
VENDOR NUMBER EQ 1892
```

This results in record 21 of the VENDORS database being qualified and selected.

At this point, there are seven lists of selected records--one list for each record in the Collect File. To summarize, the number of selected records collected from processing the seven Collect File records are as shown in this chart:

CF	CF	CF	CF	CF	CF	CF
record 1	record 2	record 3	record 4	record 5	record 6	record 7
(1)	(5)	-null-	(13)	(16)	(8)	(21)

Because Collect File items are specified in the retrieval-clause, the retrieval-clause is processed once for each Collect File record, using the list of selected records corresponding to that Collect File record. The output from the LIST command is as follows:

ORDER NUMBER	CUSTOMER NAME	PART NAME	VENDOR NAME
1	ACME TOYS	BICYCLE PEDAL	KID MANUFACTURING
1	ACME TOYS	BICYCLE PEDAL	FUN, INC.
2	ACME NUCLEAR DEVICES	PLUTONIUM	
3	ACME LAWN SUPPLIES	LAWN MOWER	U CHOP EM, INC.
3	ACME LAWN SUPPLIES	LAWN MOWER	LAWN MACHINERY, INC.
4	ACME LAWN SUPPLIES	SPRINKLER	ROTATING DEVICES, LTD.
4	ACME LAWN SUPPLIES	SPRINKLER	OSCILLATING GOODS

Note that VENDOR NAME could not be obtained for the third record in the Collect File because the VEN NUMBER value was null.

If REPEAT SUPPRESS had been in effect, then the output from the LIST command would have been as follows:

ORDER NUMBER	CUSTOMER NAME	PART NAME	VENDOR NAME
1	ACME TOYS	BICYCLE PEDAL	KID MANUFACTURING FUN, INC.
2	ACME NUCLEAR DEVICES	PLUTONIUM	
3	ACME LAWN SUPPLIES	LAWN MOWER	U CHOP EM, INC. LAWN MACHINERY, INC.
4	ACME LAWN SUPPLIES	SPRINKLER	ROTATING DEVICES, LTD. OSCILLATING GOODS

Chapter 11

Retrieving in Columnar Format: LIST

General Format 11-2

LAYOUT SPECIFICATIONS 11-4
Page Width 11-6
Title and Paging 11-6
Column Headings 11-10
Multiple-Line Headings 11-16
Column Width 11-19

THE S-SPECIFICATION 11-25
Spacing between Rows 11-28
Repeated Headings and Suppressed Row Identifiers 11-31

The LIST command displays data in a columnar format. Syntactically, the main difference between a LIST command and other retrieval commands is that the format of a LIST command provides several options, called *layout specifications*, for the layout of the output. For example, you can specify a report title, a footing line, and column headings. The layout specifications enable you to produce reports that satisfy a large number of reporting requirements.

If you do not use layout specifications, LIST automatically generates column headings and adjusts column spacing.

Except for the layout specifications and the use of records in the retrieval-clause of a LIST command, LIST and PRINT commands share the same command syntax. (As explained in the description of the syntax, records can be used only in certain cases.) Therefore, the discussion in this chapter focuses mainly on the format of LIST command layout specifications.

General Format

```
LIST [/[options] [, TITLE specifications]/] retrieval-clause
     [[,ordering-clause] where-clause] :
```

options is a list of one or more of the following format options, separated by commas. Defaults are listed in the first column.

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

See Chapter 15, "Changing Output Format: Format Options" for information about format options.

TITLE allows you to include layout specifications. Include the comma before the keyword TITLE only if you specify format specifications.

specifications are the layout specifications. See **Layout Specifications** on page 11-4 for format and explanations.

retrieval-clause specifies a list of one or more retrieval objects, separated by commas. A LIST retrieval-clause can contain a maximum of 59 retrieval objects. See **The Retrieval-Clause** on page 9-2 for details.

ordering-clause specifies the order of selected data records. See Chapter 16, "Specifying Output Order: Ordering-Clause" for more information. If you specify an ordering-clause, you must also specify a where-clause.

where-clause selects specific data records. See Chapter 24, "The Where-Clause" for details.

The retrieval-clause and where-clause for a LIST command are like those for PRINT and UNLOAD, except that in a LIST retrieval-clause, you can use record names or component numbers only in by-phrases or with the COUNT system function.

With LIST command defaults, values are displayed under their component (or Collect File) names with three spaces between columns, as shown in Example 1.

Example 1

This example shows a LIST command with no layout specifications. The column headings default to the component names. The retrieval-clause specifies an item number and an item name.

```
LIST C1, LAST NAME WHERE C1 LE 1005:

* EMPLOYEE NUMBER   LAST NAME
***
*            1001   WATERHOUSE
*            1003   SALAZAR
*            1002   BOWMAN
*            1004   KNIGHT
*            1005   KNAPP
```

LAYOUT SPECIFICATIONS

The optional layout specifications for a LIST command follow the keyword TITLE as shown below.

```
LIST [/[options] [, TITLE specifications]/] retrieval-clause
     [[,ordering-clause] where-clause] :
```

specifications [(*page-width*)] [*title*] [*paging*] [*rows*]

(*page-width*) is the maximum number of characters for each line of output, from 4 to 131. The defaults are 131 for batch mode, and 79 (for 80-character terminals) and 131 (for 132-character terminals) for interactive mode.

title specifies a report title at the top of the page using this format:

D (*starting-column*) [*title-line-text*]

paging specifies the number of lines per page and a footing line, using the following format. (Lines per page must be from 10 to 60.)

F (*lines-per-page*) [*footing-line-text*]

rows specifies column headings and row spacing. You can specify up to five rows of column headings. (Default column headings are component names or Collect File names.)

[|L (*width*)] [*column-heading*] [,*S-specification*]
  |R
  |B

L specifies a left-justified column heading.

R specifies a right-justified column heading.

B specifies a blank column; optional heading is left-justified. You can include any number of B-specifications, although no more than 59 for data columns. B-specifications can be adjacent.

(*width*) is the width of the column heading.

column-heading [[*text*] +] [[*text*] +] *text*

S-specification tells the software where to start a new row, how to space between rows of column headings, and how to space between corresponding rows of values. See **The S-Specification** on page 11-25 for format and explanations.

When you include layout specifications in a LIST command, use commas to separate the specifications.

The following symbols are keywords within the specifications that signal the end of a heading, a break in a heading line, and the end of the layout specifications, respectively:

```
,  +  /
```

These symbols cannot appear in a column heading. Be careful when including system separators in layout specifications, because the software can interpret them as string calls. The default system separator is the asterisk.

As shown in the following examples, a single line of three asterisks is displayed between the title and column headings and the data. The first column for data begins under the third asterisk.

Example 2

This example shows a LIST command with layout specifications. The command includes page width, title, paging, and column heading specifications. The italicized syntax is the column heading specification. (Details on each layout specification are discussed after this example.)

```
LIST /TITLE (45)  D(15)REPORT TITLE,  F(15)   --FOOTING LINE--,
   L(10) LAST NAME,EMPLOYEE+NUMBER,R(15)SOCIAL   +SECURITY NUMBER/
   C2, C1, C6 WHERE C1 LE 1005:
```

In the resulting output, LAST NAME, EMPLOYEE NUMBER, and SOCIAL SECURITY NUMBER make up the row of column headings, WATERHOUSE, 1001, and 654-44-6089 make up a row of values, and each column makes up a column of values.

```
                REPORT TITLE                    1
                 01/22/1990

*  LAST NAME    EMPLOYEE          SOCIAL
                 NUMBER        SECURITY NUMBER [1]
***
* WATERHOUSE        1001     654-44-6089  [2]
* SALAZAR           1003     170-83-7516
* BOWMAN            1002     329-88-6729
* KNIGHT            1004     942-62-3354
* KNAPP             1005     353-43-1272

   --FOOTING LINE--
      [3]             [3]          [3]
```

[1] Row of column headings.

[2] Row of values.

[3] Columns of values.

Page Width

Specifying page width is optional. The specification (given after the word TITLE)

(page-width)

defines the number of characters for each line of output. The page width must be at least 4 and no more than 131. If not specified, the page width defaults to 131 characters for batch output. For interactive output, the page width defaults to 79 for 80-character terminals and 131 for 132-character terminals. Page width is calculated from the asterisk that appears as the first character in a line of LIST output and therefore indicates the start of the line.

The page width must accommodate the title and footing line. Note that a page number uses five spaces.

Title and Paging

Report titles and paging are optional. The specification

D *(starting-column)* [*title-line-text*]

displays the title at the top of each new page, starting in the specified column number. The current date is placed under the title on the next line. A title requires three lines of output per page: the title, the date, and a blank line.

The specification

F *(lines-per-page)* [*footing-line-text*]

designates the number of lines per page (including title and footing) and the text for the footing line. The number of lines per page must be at least 10 but no more than 60. At least one row of values is displayed per page regardless of the paging specification.

In interactive mode, six blank lines replace a page eject. The footing line begins at the left margin. When paging is specified, a page number appears on each page. A page number uses five characters and must be accommodated in the page width. With no title specification, the page number is right-justified on the footing line. With a title, the page number is displayed right-justified on the title line. With no paging specification, output is continuous and no page number appears.

The text (including blanks) for both title and footing line begins in the first space after the right parenthesis and ends at the comma or slash that closes it. You cannot include a colon (or any command terminator), a comma, or a slash in the text for the title or footing line.

Including title or paging specifications causes each page of the report to start on a separate page, provided that RECFM=FBA is coded in the JCL DD statement for the Report File.

When the DOUBLE SPACE format option is in effect, the actual page consists of the number of lines specified. For example, the specification F(10) produces five actual lines of output (including the title, date, and footing line) double-spaced.

Syntax and semantics for titles and paging are summarized in the following illustration. Examples 3 through 6 show LIST commands with title and paging specifications.

Illus. 11.1 Title and Paging Specifications

	Result	
Specification	Page Size	Placement of Page Number
no D, no F	continuous output	not applicable
D only	continuous output	not applicable
F only with footing line text	lines-per-page	right-justified on footing line
without footing line text	lines-per-page	right-justified on blank text footing line
D and F	lines-per-page	right-justified on title line

Example 3

This LIST command includes page width and title specifications. The syntax *(45)* sets the page width to 45. The syntax *D(12)* starts REPORT TITLE in column 12.

```
LIST /TITLE (45) D(12)REPORT TITLE /C1, C6 WHERE C1 LE 1005:

          REPORT TITLE
           01/22/1990

* EMPLOYEE NUMBER   SOCIAL SECURITY NUMBER
***
*            1001   654-44-6089
*            1003   170-83-7516
*            1002   329-88-6729
*            1004   942-62-3354
*            1005   353-43-1272
```

Example 4

This LIST command includes page width and paging specifications. The page width is set to 45. Notice the page number on the footing line.

```
LIST /TITLE (45) F(15)FOOTING LINE/C1,C6 WHERE C1 LE 1010:

* EMPLOYEE NUMBER   SOCIAL SECURITY NUMBER
***
*            1001   654-44-6089
*            1003   170-83-7516
*            1002   329-88-6729
*            1006   776-94-3545
*            1004   942-62-3354
*            1007   675-29-9081
*            1009   543-87-1934
*            1008   123-12-0987
*            1010   105-32-9011
*            1005   353-43-1272

FOOTING LINE                          1
```

Example 5

This LIST command also includes page width and paging specifications. But because paging is specified to be 10 lines instead of 15, as in Example 4, two pages of output are needed for the same data. Notice that there are five spaces between the *F(10)* and *FOOTING* syntax.

```
LIST /TITLE (45) F(10)      FOOTING LINE/C1, C6 WHERE C1 LE 1010:
                                  ↑
                                5 spaces


* EMPLOYEE NUMBER   SOCIAL SECURITY NUMBER
***
*            1001   654-44-6089
*            1003   170-83-7516
*            1002   329-88-6729                    { 10 lines
*            1006   776-94-3545
*            1004   942-62-3354
*            1007   675-29-9081

     FOOTING LINE                              1


-----------------------------------------------------------------------


* EMPLOYEE NUMBER   SOCIAL SECURITY NUMBER
***
*            1009   543-87-1934
*            1008   123-12-0987
*            1010   105-32-9011
*            1005   353-43-1272

     FOOTING LINE                              2 ← page number
  ↑
5 spaces
```

Example 6

This LIST command includes page width, title, and paging specifications.

```
LIST /TITLE (45) D(12)REPORT TITLE,F(18)##FOOTING LINE##/ C1, C6

   WHERE C1 LE 1010:

          REPORT TITLE                          1 ← page number
           01/22/1990

* EMPLOYEE NUMBER   SOCIAL SECURITY NUMBER
***
*            1001   654-44-6089
*            1003   170-83-7516
*            1002   329-88-6729
*            1006   776-94-3545
*            1004   942-62-3354
*            1007   675-29-9081
*            1009   543-87-1934
*            1008   123-12-0987
*            1010   105-32-9011
*            1005   353-43-1272

##FOOTING LINE##
```

Column Headings

Column heading specifications are optional. Default column headings are the component names or Collect File names. Default headings are left-justified for text items and right-justified for numeric items and dates.

When the STUB format option is in effect, default headings are displayed if neither a column width nor a column heading is specified. The STUB SUPPRESS format option prevents the display of default column headings but does not affect specified headings.

If you do not specify column headings, the software chooses the larger of the number of characters in the default column headings and the default column widths to determine the column width.

When a default column heading is longer than 50 characters, the first 50 characters are written on one line, the next 50 on the next line, and another 50 on a third line. Any characters after 150 are discarded. Default column headings are summarized in the following illustration and in Example 7.

Illus. 11.2 Default Column Headings

Retrieval-Clause Object	Default Heading
Components	the component name
System Functions	the function name
Ad hoc Functions	none
Collect File items	the Collect File item name

Example 7

This example shows default column headings for an item, a stored function, an ad hoc function, and a system function. The LIST command is preceded by two DESCRIBE commands to display the definitions of C2001 and C111.

```
DESCRIBE C2001:  DESCRIBE C111:

2001*  AGE (INTEGER FUNCTION ( ((*FTODAY* - BIRTHDAY)/365.25)))
     111*  PAY RATE (MONEY $9999.99 IN 110)


LIST LAST NAME, *C2001*, (C106 - C105), AVG C111 WHERE C106 EXISTS:

* LAST NAME            AGE                 AVG PAY RATE
***
* REID                  33            378       $2,325.00
* SALAZAR               37            367
* GARRETT               43            729
* GARRETT               43            365
* SCHOLL                33            302
* JUAREZ                31            243
* AMEER                 27            578
```

You can specify up to three lines of column headings, using the following format:

```
[ |L (width) ] [column-heading]
  |R
  |B
```

The column heading text begins in the space following the right parenthesis and ends at the comma or slash that closes it. Blanks within the heading are retained. The heading can contain any symbols except a colon, a comma, a slash, or a plus.

L specifies that the heading is to be left-justified. R specifies a right-justified heading. B specifies a column with no data; if a heading is specified, it is left-justified.

The *width* specifies the width of the column. If the text contains more than *width* characters, the column is expanded to contain the entire heading. For instance, see the column heading specification L(1) CITY AND STATE in Example 24.

The total width of the columns in any row must be at least two spaces less than the page width in order to contain row identifiers and space at the left of each row.

If columns of data are specified with no B-specification in between, the software automatically generates three spaces between the data columns. Blanks between column heading specifications (that is, between the comma and the following column heading specification) are ignored. For example, in the following command, the blanks between the comma and R(15) are not included in either the first or second headings.

```
... L(15)FIRST HEADING,          R(15) SECOND HEADING ...
```

When a column heading specification does not include text for the column heading, the report has a blank heading for that column. When a column heading specification does not include an L-, R-, or B-specification, the column heading is left-justified, and the software allots the number of characters in the largest of the three lines as the number determining the column width. For example, notice how SOCIAL is left-justified in the output of the second command in Example 13.

Blanks between column specifications (that is, between the comma and the next column specification) are ignored unless that next column specification does not begin with an L, an R, or a B. In that case, the blanks after the comma become part of the text of that next column specification. Therefore, if the column specifications do not fit on one line, complete the last column specification that fits on one line with a comma and begin the next column specification with an L, an R, or a B on the next line.

Column heading specifications, except those with a B-specification for blank columns, correspond one-to-one with the objects in the retrieval-clause. If you supply fewer column heading specifications than retrieval objects, the software uses default headings for the rightmost objects. This correspondence is shown in the following command:

```
LIST/TITLE FIRST, B(5)BLANK, L(6)SECOND,B(4),R(6)THIRD/
   item1, item2, item3, item4 WHERE item1 EXISTS:
```

In the resulting report, FIRST would appear above the values for *item1*, SECOND above the values for *item2*, THIRD above the values for *item3*, and ITEM4 (the default heading) above the values for *item4*.

The format for column heading specifications and the resulting output are summarized in the following illustration and shown in Examples 8 through 11.

Illus. 11.3 Column Specifications

Specification	Display
None	Depends on retrieval object Object — Heading Component — component name System Function — function name Ad Hoc Function — none Collect File Item — Collect File item name (Default headings for CHARACTER, TEXT and UNDEFINED items are left-justified. Default headings for dates and numeric items are right-justified.)
,bXXX XXXb,	bXXX XXXb — bXXX XXXb
|L (width), |R |B	None (used to define position of data and column width)
|L (width)XXX XXX, XXX XXX |R |B	(justified according to the specification, left-justified, if B)

Example 8

In this example, the text for two of the column headings is specified. EMPLOYEE STATUS is the default heading for C9, the rightmost item in the retrieval-clause.

```
LIST /TITLE EMP-NUMB,ETHNIC ORIGIN/ C1, C8, C9 WHERE C1 =< 1010:

* EMP-NUMB   ETHNIC ORIGIN   EMPLOYEE STATUS
***
*     1001   CAUCASIAN       FULL TIME
*     1003   HISPANIC        FULL TIME
*     1002   CAUCASIAN       FULL TIME
*     1006   CAUCASIAN       FULL TIME
*     1004   CAUCASIAN       FULL TIME
*     1007   BLACK           FULL TIME
*     1009   CAUCASIAN       FULL TIME
*     1008   HISPANIC        FULL TIME
*     1010   CAUCASIAN       FULL TIME
*     1005   CAUCASIAN       FULL TIME
```

Example 9

In this example, both column specifications include column width and text. The column heading MAJOR FIELD appears right-justified, and MINOR FIELD appears left-justified. Notice that in the output there are three spaces between the end of the MAJOR FIELD column and the beginning of the MINOR FIELD column. Also, there are sixteen spaces in the MINOR FIELD column.

```
LIST /TITLE R(16)MAJOR FIELD, L(16)MINOR FIELD/ C414, C415

   WHERE C415 EXISTS & C1 LE 1100:
```

3 spaces ↓ *16 spaces* ↓

```
*      MAJOR FIELD   MINOR FIELD
***
* ENGLISH            ART
* MATHEMATICS        BUSINESS ADMINIS
                     TRATION
* EDUCATION          HISTORY
* PSYCHOLOGY         MARKETING
* MATHEMATICS        BUSINESS ACCOUNT
                     ING
* COMPUTER SCIENCE   MATHEMATICS
* MARKETING          ENGLISH
* EDUCATION          PHYSICS
* COMPUTER TECHNOL   FINANCE
  OGY
* BUSINESS ADMINIS   PSYCHOLOGY
  TRATION
* ECONOMICS          MATHEMATICS
```

Example 10

This example is the same LIST command as Example 9, except that in this example, the column heading MAJOR FIELD appears left-justified and MINOR FIELD appears right-justified.

```
LIST /TITLE L(16)MAJOR FIELD, R(16)MINOR FIELD/ C414, C415

   WHERE C415 EXISTS & C1 LE 1100:

* MAJOR FIELD             MINOR FIELD
***
* ENGLISH                 ART
* MATHEMATICS             BUSINESS ADMINIS
                          TRATION
* EDUCATION               HISTORY
* PSYCHOLOGY              MARKETING
* MATHEMATICS             BUSINESS ACCOUNT
                          ING
* COMPUTER SCIENCE        MATHEMATICS
* MARKETING               ENGLISH
* EDUCATION               PHYSICS
* COMPUTER TECHNOL        FINANCE
  OGY
* BUSINESS ADMINIS        PSYCHOLOGY
  TRATION
* ECONOMICS               MATHEMATICS
```

Example 11

In this example, the first column specification includes column width and text, the second column specification includes column width and text for a column without data, and the third column specification includes column width but no text. The number 12 in the B-specification causes twelve blank spaces to be generated between the two data columns instead of the three spaces generated by default.

```
LIST /TITLE L(16)THIS EMPLOYEE, B(12)WAS HIRED ON,L(10)/

   C2, C4 WHERE C1 LE 1010:

* THIS EMPLOYEE   WAS HIRED ON
***
* WATERHOUSE                 01/01/1988
* SALAZAR                    01/01/1988
* BOWMAN                     01/01/1988
* GARRETT                    01/01/1988
* KNIGHT                     01/01/1988
* BROWN                      01/01/1988
* JONES                      04/01/1988
* HERNANDEZ                  01/01/1988
* SMITH                      01/01/1988
* KNAPP                      01/01/1988
---------------- ------------ ----------
     ↑               ↑           ↑
 16 spaces       12 spaces    10 spaces
```

Multiple-Line Headings

Column headings can have one to three lines. The symbol + designates a line break. Text following the plus is displayed on the next line. If no text precedes a single plus, one line is left blank. Two consecutive plus signs produce two blank lines.

To obtain a report with two-line column headings, for example,

Line 1	text1	text2	
Line 2	text1		text3

specify the following for each heading, respectively:

```
text1 + text1
text2
+ text3
```

To obtain a report with three-line headings, for example,

Line 1	text1	text2	text3		text5		
Line 2	text1	text2		text4		text6	
Line 3	text1		text3	text4			text7

specify the following for each heading, respectively:

```
text1 + text1 + text1
text2 + text2
text3 + + text3
+ text4 + text4
text5
+ text6
+ + text7
```

Examples 12, 13, and 14 show multiple-line headings.

Example 12

This LIST command specifies a one-line heading.

```
LIST/TITLE EMPLOYEE NUMBER,LAST NAME,SOCIAL SECURITY NUMBER/

   C1, C2, C6 WHERE C1 EQ 1265:

* EMPLOYEE NUMBER   LAST NAME   SOCIAL SECURITY NUMBER
***
*          1265    SLYE        434-21-1300
```

Example 13

This example shows two LIST commands that specify two-line column headings.

```
LIST/TITLE EMPLOYEE+NUMBER,LAST NAME,SOCIAL SECURITY NUMBER/

   C1, C2, C6 WHERE C1 = 1265:

  * EMPLOYEE    LAST NAME   SOCIAL SECURITY NUMBER
    NUMBER
  ***
  *     1265    SLYE        434-21-1300
```

```
LIST/TITLE EMPLOYEE+NUMBER,+LAST NAME,SOCIAL+SECURITY NUMBER/

   C1, C2, C6 WHERE C1 = 1265:

* EMPLOYEE              SOCIAL
  NUMBER     LAST NAME  SECURITY NUMBER
***
*     1265   SLYE       434-21-1300
```

Example 14

This example shows three LIST commands that specify three-line column headings.

```
LIST/TITLE EMPLOYEE NUMBER,+LAST NAME,SOCIAL+SECURITY+NUMBER/

   C1, C2, C6 WHERE C1 EQ 1265:

* EMPLOYEE NUMBER              SOCIAL
                    LAST NAME  SECURITY
                               NUMBER
***
*           1265   SLYE       434-21-1300


LIST/TITLE EMPLOYEE+NUMBER,LAST++NAME,SOCIAL+SECURITY+NUMBER/

   C1, C2, C6 WHERE C1 EQ 1265:

* EMPLOYEE   LAST   SOCIAL
  NUMBER           SECURITY
             NAME   NUMBER
***
*     1265   SLYE   434-21-1300


LIST/TITLE +EMPLOYEE+NUMBER,++LAST NAME,SOCIAL+SECURITY+NUMBER/

   C1, C2, C6 WHERE C1 EQ 1265:

*                        SOCIAL
  EMPLOYEE               SECURITY
  NUMBER     LAST NAME   NUMBER
***
*     1265   SLYE        434-21-1300
```

Column Width

You can either specify a column width or use the default. Defaults apply when neither a heading nor a column width is specified. The actual column width corresponds to the larger of the retrieval object's name or default column width. Default column widths are shown in Examples 15 and 16.

Determinants of column width are summarized in the following illustration. Default column widths are shown in Illus. 11.5 on page 11-22. Default column headings are shown in Illus. 11.2 on page 11-11.

Illus. 11.4 Actual Column Width for LIST Output

Context	Actual Width
Default (No Heading or Column Width Specified):	
Default heading greater than default width	Default heading
Default width greater than default heading	Default width
Specification Included:	
Column heading, no specified width	Column heading
Column heading greater than specified width	Column heading
Specified width, no heading specified	Specified width
Specified width greater than specified heading	Specified width

If you specify either a column heading or a column width, that specification determines the column width. If both a heading and a width are specified, the larger of the two determines the actual column width, as shown in Examples 17 and 18.

When no blank columns (B-specifications) are specified, the software allows three spaces between columns.

Example 15

In this example, the number of characters in the retrieval-clause item names is larger than the default width; thus, the item name determines the column width. The DESCRIBE command displays the definition of the items.

```
DESCRIBE C1 THROUGH C5:

1*  CHAR ITEM (CHAR X(5))
2*  DATE TYPE ITEM (DATE)
3*  INT ITEM (INTEGER NUMBER 9999)
4*  DECIMAL ITEM (DECIMAL NUMBER 9999.99)
5*  MONEY ITEM (MONEY $9999.99)

LIST C1, C2, C3, C4, C5:

* CHAR ITEM   DATE TYPE ITEM   INT ITEM   DECIMAL ITEM   MONEY ITEM
***
* AAAAA           01/01/1990       1234        1234.56    $1,234.56
```

Example 16

In this example, the number of characters in the retrieval-clause item names is less than the default width; thus, the default width determines the column width. The DESCRIBE command displays the definition of the items.

```
DESCRIBE C6 THROUGH C10:

 6*  CHAR (CHAR X(5))
 7*  DATE (DATE)
 8*  INT (INTEGER NUMBER 9999)
 9*  DEC (DECIMAL NUMBER 9999.99)
10*  MON (MONEY $9999.99)

   LIST C6, C7, C8, C9, C10:

* CHAR          DATE       INT        DEC          MON
***
* AAAAA   01/01/1991      1234    1234.56   $1,234.56
```

1 **2** **3** **4** **5**

1 CHARACTER item, picture X(5), default width = 5.

2 DATE item, default width = 10.

3 INTEGER item, picture 9(4), default width = 4+2 = 6.

4 DECIMAL item, picture 9(4).9(2), default width = 4+2+3 = 9.

5 MONEY item, picture 9(4).9(2), default width = 4+2+4 = 10.

Example 17

In this example, the number of characters in the text of the first column exceeds the column width specified; thus, the column width is determined by the number of characters in FIRST NAMES, that is, 11. The number of characters in LAST NAMES is less than 20, hence 20 is the column width of the second column. Notice that LAST NAMES appears right-justified.

```
LIST /TITLE L(6)FIRST NAMES, R(20)LAST NAMES/ C3, C2 WHERE C1 LE 1010:
```

```
          3 spaces   20 spaces
             ↓          ↓
* FIRST NAMES            LAST NAMES
***
* CLIFTON P.    WATERHOUSE
* YOLANDA       SALAZAR
* HUGH E.       BOWMAN
* OLAN M.       GARRETT
* ALTHEA        KNIGHT
* VIRGINA P.    BROWN
* MICHAEL Y.    JONES
* JESSE L.      HERNANDEZ
* JANET F.      SMITH
* PATRICE R.    KNAPP
```

Example 18

In this example, the number of spaces between the two columns of data is reduced to one by specifying B(1).

```
LIST /TITLE L(6)FIRST NAMES,B(1), R(16)LAST NAMES/ C3, C2 WH C1 <= 1010:
```

```
           1 space
              ↓
* FIRST NAMES        LAST NAMES
***
* CLIFTON P.   WATERHOUSE
* YOLANDA      SALAZAR
* HUGH E.      BOWMAN
* OLAN M.      GARRETT
* ALTHEA       KNIGHT
* VIRGINA P.   BROWN
* MICHAEL Y.   JONES
* JESSE L.     HERNANDEZ
* JANET F.     SMITH
* PATRICE R.   KNAPP
```

Illus. 11.5 Default Column Widths for LIST Display

Retrieval-Clause Object	Picture	Default Width
Components		
Items		
CHAR, TEXT, or UNDEFINED	X(n)	n
DATE		10
INTEGER	9(n)	n + 2
DECIMAL	9(m).9(n)	m + n + 3
MONEY†	$9(m).9(n)	m + n + 4
REAL		14
DOUBLE		24
Stored Functions		
DATE		10
INTEGER		11
DECIMAL		12
MONEY		13
REAL		14
DOUBLE		24
System Functions		
COUNT *component*		10
SUM *item*	††	††
MIN *item*	††	††
MAX *item*	††	††
AVG *item*	††	††
SIGMA *item*	††	††
system-function adhoc-function		†††
Ad hoc Functions	none	24

Notes:

† Money values are displayed with commas.

†† MIN, MAX, SUM, AVG, and SIGMA use the picture of the item specified or 14 characters for REAL and 24 characters for DOUBLE.

††† Depends on the highest type in the ad hoc function, that is, 13 for DECIMAL, MONEY, or INTEGER and 14 for REAL and 24 for DOUBLE.

If the actual column width is inadequate for the values (that is, if a value has more characters than the column width allows), then

- textual values wrap within the column to a total of five lines. If five lines cannot accommodate the entire value, an asterisk is displayed as the last character on the fifth line to indicate truncation of the value.

- dates and numeric values are truncated on the left (an asterisk appears as the first character) if the actual width (see Illus. 11.4 on page 11-19) is more than the default width (see Illus. 11.5 on page 11-22). Dates and numeric values do not extend beyond the first line.

The CR for negative MONEY values is displayed to the right of the value. If the next column has a B(1) specification, only the C appears. Similarly, B(0) would preclude any indication that a MONEY value is negative. When fewer than two blanks are available, the CR is written over by the following field. However, when a column of MONEY values is last on its row, the CR appears even if it means exceeding a specified page width.

REAL and DOUBLE values are displayed in floating point format with precision based on the number of characters in the column heading. The minimum column width without truncation is 8 characters. The maximum data widths are 14 characters for REAL and 24 characters for DOUBLE, right-justified. The value appears in E-notation, with precision based on the number of spaces available in the column heading (if specified) or a default width of 14 for REAL and 24 for DOUBLE. The format is ±n.nE±ee to ±n.nnnnnnnnnnnnnnnnE±ee, depending on display space available under the column heading. The exponent E±ee is always a two-digit signed integer. If the column width for an item is too narrow, truncation occurs to the least significant digits of the floating point characteristic, and an asterisk at the beginning of the value indicates the truncation.

You can choose the output format for UNDEFINED items with the HEX OFF/HEX ON format option. You can set this option with the FORMOP command or any retrieval command, which is similar to specifying other format options.

HEX OFF (the default) means display format. Any unprintable characters in the LIST output are displayed as periods and message -304- appears. HEX ON means HEX notation format. A missing value (null) is ed as displayed as -NULL- for either format.

The following rules apply when the LIST output is being formatted:

- The LIST command fills the specified columns for an UNDEFINED item with as many characters as possible before truncating. However, if the system truncates only trailing binary zeros, it does not display the asterisk that normally indicates truncation.

- For HEX ON, the list command requires a minimum of five columns to contain the word HEX, a blank, and a minimum one-character item name in the column heading. For HEX ON, an even number of hexadecimal characters are displayed. Characters beyond the 125th byte (250 hexadecimal characters) are truncated.

- For HEX OFF, the LIST command does not truncate any data that fit within the specified number of columns for the UNDEFINED item.

When a value that is not defined results from calculations for system, stored, and ad hoc functions, X's are displayed where the value would appear. for example, division by zero.

Examples 19 and 20 show the continuation of textual values, truncation of values, and display of values that are not defined.

STUB SUPPRESS governs default column headings, but it does not affect specified headings. When a default heading is suppressed, the default width alone determines the column width. The column width for ad hoc functions remains 13.

Example 19

This example shows how textual values can extend to up to five lines and how a system separator appears to the left of truncated numeric items and dates. A PRINT command is shown for comparison.

```
PRINT /NAME/ C1, C2, C3, C4, C5:

ITEM 1* TEXTUAL VALUES EXTEND TO FIVE LINES
ITEM 2*  1234
ITEM 3* 12/31/1991
ITEM 4*      $123.45CR
ITEM 5*   $1,234.56


LIST /TITLE R(7)ITEM 1,ITEM+2, ITEM 3,ITEM 4, ITEM 5/C1,C2,C3,C4,C5:

*  ITEM 1    ITEM     ITEM 3    ITEM 4    ITEM 5
             2
***
* TEXTUAL    1234     */1991    *23.45CR  *34.56
   VALUES
   EXTEND
   TO FIV
  E LINES
```

Example 20

This example shows a truncated textual value (the system separator appears in the fifth line) and the X's displayed for a value that is not defined. Notice that because one column was specified after ITEM 4, only the C of CR appears. Had the specification been B(0), neither the C nor the R would have appeared.

```
LIST /TITLE L(10)ITEM 4,B(1),R(4)ITEM+ 1,UNDEFINED+  VALUE/

   C4,C1,(C5/(1234 - C2)):

* ITEM 4     ITEM   UNDEFINED
              1      VALUE
***
*    $123.45CTEXT   XXXXXXXXX
          UAL
          VALU
          ES E
          XTE*
```

THE S-SPECIFICATION

The S-specification in a LIST command tells the software where to start a new row, how to space between rows of column headings, and how to space between corresponding rows of values.

The S-specification is part of a LIST command's layout specifications:

[(page-width)] [title] [paging] [*rows*]

---

rows specifies column headings and row spacing, using this format:

```
[ |L  (width)] [column-heading]  [,S-specification]
  |R
  |B
```

S-specification

```
S |(i)
  |(h [,v])
  |(h [,v, h-list])
```

i is a number denoting spacing after a row of headings and corresponding rows of values.

h is a number denoting spacing after a previous row of headings. It must be one or greater.

v is a number denoting spacing after each row of data corresponding to the preceding row of headings. It must be one or greater.

h-list

```
c [,c] ...
```

c

```
|row-number
|P
|N
```

row-number is a number from one to five, inclusive, that identifies the row of headings to be displayed above each row of data corresponding to the preceding row of headings.

P displays column headings following the title line and also as specified in each S-specification. You can use P only in the first S-specification.

N suppresses display of all headings. You can use N only in the first S-specification.

The LIST commands in Examples 21 and 22 include S-specifications.

Example 21

In this LIST command, S-specifications are included to separate rows and to designate spacing between rows. The first italicized syntax is the first row specification, and the second italicized syntax is the second row specification.

```
LIST /TITLE (40)  D(7)REPORT TITLE,  F(40)      --FOOTING LINE--,
   L(10) LAST NAME,S(1,2),B(10),
   L(8)EMPLOYEE+ NUMBER,R(15)SOCIAL    +SECURITY NUMBER,S(2)/
   C2, C1, C6 WHERE C1 LE 1005:

     REPORT TITLE                    1
      01/22/1990

*  LAST NAME  1
-           EMPLOYEE        SOCIAL
             NUMBER    SECURITY NUMBER  2

***
* WATERHOUSE

-              1001   654-44-6089

* SALAZAR

-              1003   170-83-7516

* BOWMAN

-              1002   329-88-6729

* KNIGHT

-              1004   942-62-3354

* KNAPP

-              1005   353-43-1272

     --FOOTING LINE--
```

1 First row of column headings.

2 Second row of column headings.

Example 22

In this LIST command, S-specifications place column headings immediately above their corresponding values. The first italicized syntax is the first row specification, and the second italicized syntax is the second row specification.

```
LIST /TITLE (30)  D(7)REPORT TITLE,  F(60)     --FOOTING LINE--,
   L(10) LAST NAME,S(1,2,1),
   L(8)EMPLOYEE+ NUMBER,R(15)SOCIAL    +SECURITY NUMBER,S(2,2,2)/
   C2, C1, C6, ORDERED BY C2 WHERE C1 LE 1003:

        REPORT TITLE          1
         01/22/1990

***
 LAST NAME [1]
BOWMAN

 EMPLOYEE       SOCIAL      [2]
 NUMBER     SECURITY NUMBER

    1002    329-88-6729

 LAST NAME [1]
SALAZAR

 EMPLOYEE       SOCIAL      [2]
 NUMBER     SECURITY NUMBER

    1003    170-83-7516

 LAST NAME [1]
WATERHOUSE

 EMPLOYEE       SOCIAL      [2]
 NUMBER     SECURITY NUMBER

    1001    654-44-6089


  --FOOTING LINE--
```

[1] First row of column headings.

[2] Second row of column headings.

Spacing between Rows

When the page width is not large enough to hold all the columns, LIST begins a new row after the last full column that fits across the page. You, however, can specify when a new row is to begin by using an S-specification.

An S-specification indicates the end of a row and causes the next column specified to begin in column one. You can specify up to five rows of columns; that is, a LIST command can include no more than five S-specifications. An S-specification cannot appear by itself; it must follow at least one column specification.

Each row is identified by a symbol. An asterisk (*) appears in column 1 of the first line of the first row of column headings and in column 1 of the first line of values corresponding to those headings. A minus sign (-) identifies the second row; an equals sign (=) identifies the third row; a plus sign (+) identifies the fourth row; and a slash (/) identifies the fifth row.

Note that changing the system separator from the asterisk does not change the asterisks appearing before the first row of values or the asterisks serving as the first row marks.

You can also use the S-specification to control spacing between rows of column headings, as well as between corresponding rows of values. Otherwise, by default, LIST output is single-spaced.

Using the S-specification to control spacing consists of the following format:

```
S |(i)
  |(h [,v])
```

where

- *i* is a number denoting spacing after the row of headings and corresponding rows of values.
- *h* is a number denoting spacing after a previous row of headings. It must be one or greater.
- *v* is a number denoting spacing after each row of data corresponding to the preceding row of headings. It must be one or greater.

For example, if you specify S(2), the software double-spaces after the row of headings and after each row of corresponding values.

With the DOUBLE SPACE format option in effect, the software double-spaces for each line. Therefore, the following specification leaves three blank lines between rows:

```
LIST/DOUBLE SPACE, TITLE columns S(2,2)/ ...
```

And this specification leaves one blank line between rows:

```
LIST/SINGLE SPACE, TITLE columns S(2,2)/ ...
```

Example 23

This example shows two LIST commands. The first command does not include any S-specifications. Because a page width of 30 is specified, there is no space for the values for GENDER to fit in the first row; therefore, they are placed in the second row.

The second LIST command specifies the same column headings as the first command; however, it includes four S-specifications to separate rows and to direct spacing between rows. The last row has no S-specification. Therefore, the software single-spaces after the heading and the corresponding values. If only one number is specified (for example, S(2)), the number indicates spacing for both column headings and values. Notice, also, the five row identifiers on the left margin of the output; the row specifications were placed one to a line in the second situation in this example. (This was done for ease of reading; it is not required.)

```
LIST /TITLE (30) L(12)LAST NAME, L(10)HIRE DATE,R(6)GENDER,

   L(15)CITY AND STATE,ZIP CODE/

   C2, C4, C7, C15, C16 WHERE C1 LE 1003:


* LAST NAME       HIRE DATE
-    GENDER    CITY AND STATE
= ZIP CODE
***
* WATERHOUSE      01/01/1988
- MALE       AUSTIN TX
= 78752
* SALAZAR         01/01/1988
- FEMALE    LEANDER TX
= 78641
* BOWMAN          01/01/1988
- MALE       AUSTIN TX
= 78741
```

(continued next page)

Example 23 (continued)

```
LIST /TITLE (30) L(12)LAST NAME,S(1),

                 L(10)HIRE DATE,S(2),
                                 1 2
                 R(6)GENDER ,S(1,2),
                                      3 4
                 L(15)CITY AND STATE,S(2,1),

           B(10),L(8)ZIP CODE/

 C2, C4, C7, C15, C16 WHERE C1 LE 1003:

    * LAST NAME
    - HIRE DATE

    =   GENDER      1
    + CITY AND STATE
                    3
    /           ZIP CODE
    ***
    * WATERHOUSE
    - 01/01/1988

    = MALE
              2
    + AUSTIN TX      4
    /           78752
    * SALAZAR
    - 01/01/1988

    = FEMALE

    + LEANDER TX
    /           78641
    * BOWMAN
    - 01/01/1988

    = MALE

    + AUSTIN TX
```

1 Single Spacing

2 Double Spacing

3 Double Spacing

4 Single Spacing

Repeated Headings and Suppressed Row Identifiers

By default, column headings are displayed below the report title. You can, however, use the S-specification to display the column headings immediately above sets of values or above individual rows of values.

In this case, the format for the S-specification is as follows:

```
S (h [,v, h-list] )
```

where

h is a number denoting spacing after a previous row of headings. It must be one or greater.

v is a number denoting spacing after each row of data corresponding to a previous row of headings. It must be one or greater.

h-list *c* [,*c*] ...

c |*row-number*
|P
|N

row-number is a number from one to five, inclusive, that identifies the row of headings to be displayed above each row of corresponding data.

P displays column headings following the title line and also as specified in each S-specification. You can use P only in the first S-specification.

N suppresses display of all headings. You can use N only in the first S-specification.

P and N are invalid except in the S-specification for the first row, and they must follow the two numbers specifying spacing in an S-specification. When both N and P are specified, the second specification governs.

Each row number identifies a row of column headings. These headings are displayed above the rows of values corresponding to the row of column specifications that precede the S-specification. The *h-list* specifies where the software is to repeat or suppress column headings. For example, assume three rows are specified, and each row consists of one or more column specifications.

```
LIST /TITLE ... row1,S(1,1,1,2,3), row2,S(1,1,2), row3,S(1)/ ...
```

- The syntax *row1,S(1,1,1,2,3)* causes the headings in row1, row2, and row3 to be displayed above each row of values corresponding to row1.
- The syntax *row2,S(1,1,2)* causes the headings in row2 to be displayed above each row of values corresponding to row2.
- The syntax *row3,S(1)* causes each row of values corresponding to row3 to be displayed without headings directly below each row of values corresponding to row2.

Notice that single spacing is specified for all rows.

While you can list the row numbers in any order, the rows of column headings are always displayed in numerical order. For example, *row1, S(1,1,1,2,3)* or *row1, S(1,1,2,1,3)* produces the following output:

```
    row1 column headings
    row2 column headings
    row3 column headings
row of values corresponding to row1
```

Since you can specify up to five rows, there can be no more than five row numbers, that is, the numbers 1, 2, 3, 4, and 5. The row numbers, if specified, must always follow the two numbers *h and v* specifying spacing. Repeating a row number does not affect output, unless repetition results in more than five row numbers being specified, for example, *S(1,1,1,2,3)*, *S(1,1,1,2,1,2,3)*, *S(1,1,1,2,3,3)* all produce the same effect. But *S(1,1,1,1,2,2,3,3)* is invalid and gives a syntax error.

When the S-specification for a given row includes row numbers, the headings specified appear only over the data corresponding to that row. To have headings above data corresponding to any other row, you must specify row number in the S-specification for that row. Examples 22, 24, and 25 show S-specifications with row numbers.

When you specify a row number, column headings appear only above rows of values and not above the three asterisks at the top of the report. To have column headings also appear at the top of the page above the three asterisks, you can specify a P in the S-specification for the first row.

To cancel all column headings in a report (for example, to have only values displayed), you can specify an N in the S-specification for the first row.

When you use an *h-list*, the row identifiers (*, -, =, +, /) are not displayed. Examples 26 and 27 use P and N specifications.

Example 24

This example shows a LIST command with three S-specifications. They specify spacing and placement of column headings immediately above their respective values. The specification *S(1,2,1)*, after the first row of column specifications tells the software to perform these tasks in order:

1. to single space, S(**1**,2,1), after it displays LAST NAME
2. to double space, S(1,**2**,1), after each value corresponding to L(10) LAST NAME
3. to place the heading, LAST NAME, specified in the first row, S(1,2,**1**), in the line immediately above each value for C2.

In the third line of the command, the column width is determined by the number of characters in CITY AND STATE, not by the L-specification. The L-specification in the third line is included in order to place it on a separate line. If L, R, or B were not included, the software would incorporate the blanks.

```
                  1
LIST /TITLE L(10)LAST NAME, S(1,2,1),

                  1
            L(10)HIRE DATE ,S(1,2,2),

                                          2
            L(1)CITY AND STATE,ZIP CODE,S(1,2,3)/
```

C2, C4, C15, C16 WHERE C1 LE 1002:

```
***
          1
  LAST NAME
  WATERHOUSE

          1
  HIRE DATE
  01/01/1988

                        2
  CITY AND STATE   ZIP CODE
  AUSTIN TX        78752

  LAST NAME
  BOWMAN

  HIRE DATE
  01/01/1988

                        2
  CITY AND STATE   ZIP CODE
  AUSTIN TX        78741
```

1 10 spaces.

2 The "3" displays CITY AND STATE and ZIP CODE over values for C15 and C16.

Example 25

This LIST command contains four S-specifications. The first and third S-specifications set spacing only. The S-specification for the second row directs the software to single space after the headings and corresponding values, S(**1,1**,1,2). Also, S(1,1,1,**2**) specifies that the heading in the first row, GENERAL INFORMATION-, and the headings in the second row are to be placed immediately above the values corresponding to the second row.

The S-specification for the fourth row directs the software to single space, S(**1**,2,3,4), after the headings, and to double space, S(1,**2**,3,4), after the corresponding values. Also, S(1,2,**3,4**), specifies that the heading in the third row, EDUCATION-, and the headings in the fourth row are to be placed immediately above the values corresponding to the fourth row.

```
      B(15),L(12)LAST NAME,SOCIAL SECURITY NUMBER,S(1,1,1,2),

      B(15)      EDUCATION-,S(2),

      B(10),L(34)SCHOOL,DEGREES,S(1,2,3,4)/

C2, C6, C411, C412 WHERE C1 SPANS 1006*1007:

***
  GENERAL INFORMATION-

              LAST NAME      SOCIAL SECURITY NUMBER
              GARRETT         776-94-3545
      EDUCATION-

           SCHOOL                                DEGREES
           TEXAS CHRISTIAN UNIVERSITY            BS

      EDUCATION-

           SCHOOL                                DEGREES
           UNIVERSITY OF WASHINGTON              MS

  GENERAL INFORMATION-

              LAST NAME      SOCIAL SECURITY NUMBER
              BROWN           675-29-9081
      EDUCATION-

           SCHOOL                                DEGREES
           UNIVERSITY OF WISCONSIN               BA
```

Example 26

This LIST command has three S-specifications. Using P in the first S-specification causes the column headings to appear immediately above both the three asterisks and the values.

```
LIST /TITLE D(15)MAILING LABELS,

        L(12)FORENAME,B(0),L(10)LAST NAME,S(1,1,1,P),

        L(25)STREET ADDRESS,S(1,1,2),

        L(10)CITY-STATE,B(3),L(5)ZIP CODE,S(1,4,3)/

C3, C2, C14, C15, C16, ORDERED BY C2 WHERE C1 LE 1003:
```

```
                MAILING LABELS
                  03/08/1990

   FORENAME    LAST NAME
   STREET ADDRESS
   CITY-STATE   ZIP CODE
 ***
   FORENAME    LAST NAME
   HUGH E.     BOWMAN
   STREET ADDRESS
   47 CYPRESS POINT CIRCLE
   CITY-STATE   ZIP CODE
   AUSTIN TX    78741


   FORENAME    LAST NAME
   YOLANDA     SALAZAR
   STREET ADDRESS
   6811 PICKET FENCE DRIVE
   CITY-STATE   ZIP CODE
   LEANDER TX   78641


   FORENAME    LAST NAME
   CLIFTON P.  WATERHOUSE
   STREET ADDRESS
   505 CAT MOUNTAIN TRAIL
   CITY-STATE   ZIP CODE
   AUSTIN TX    78752
```

Example 27

This LIST command is the same as the one in Example 26, except that N is used instead of P. Notice that this causes the suppression of all column headings.

```
LIST /TITLE D(15)MAILING LABELS,

   L(12)FORENAME,B(0),L(10)LAST NAME,S(1,1,1,N),

   L(25)STREET ADDRESS,S(1,1,2),

   L(10)CITY-STATE,B(3),L(5)ZIP CODE,S(1,4,3)/

   C3, C2, C14, C15, C16, ORDERED BY C2 WHERE C1 LE 1003:
```

As a matter of fact, the command

```
LIST /TITLE D(15)MAILING LABELS,L(12),B(1),L(10),S(1,N),

   L(25),S(1),L(15),B(3),L(5),S(1,4)/C3,C2,C14,C15,C16,

   OB C2 WHERE C1 LE 1003:
```

produces the same result, even though it does not specify text for column headings. The N is needed to suppress the display of the column identifier signs.

```
                MAILING LABELS
                  03/08/1990

***
  HUGH E.       BOWMAN
  47 CYPRESS POINT CIRCLE
  AUSTIN TX     78741


  YOLANDA       SALAZAR
  6811 PICKET FENCE DRIVE
  LEANDER TX    78641


  CLIFTON P.    WATERHOUSE
  505 CAT MOUNTAIN TRAIL
  AUSTIN TX     78752
```

Chapter 12

Retrieving in Sequential Format: PRINT

Format 12-2
Considerations 12-3
Examples 12-3

The PRINT command retrieves data from a database and/or a Collect File and displays the data in a simple, sequential list. With the PRINT command, you can

- retrieve values from any number of specified database components and/or Collect File items
- display values for stored strings and functions
- display values of ad hoc and system functions
- use a where-clause (optional)
- produce output in logical order when no where-clause is used
- produce output in a specific order with an ordering-clause when a where-clause is used
- specify format options.

The PRINT command displays data in a simple list, one value to a line. When an item, stored function, system function, or ad hoc function is specified in the retrieval-clause, the software displays values for the item or functions.

If you specify a schema record in a retrieval-clause, the software displays all items for the selected data records. Unless the RECORD format option has been specified, all items that are descendants of the selected data records are also displayed. Thus, if a record is specified, the software displays data trees having selected data records at the top.

When you issue a PRINT command without a where-clause, output is in logical order. That is, the output represents the logical order of the records from your point of view rather than the physical order of the records as stored in the database tables. For a Collect File, the values are displayed according to the sequential order of the Collect File records.

Data records are displayed in logical order regardless of the order in which they are specified in the retrieval-clause. However, you can modify output order by specifying records in by-clauses. Values for items within a data record are displayed in the order you specify.

When you issue a PRINT command with a where-clause, output is displayed in physical order. If values from disjoint records are to be displayed, a by-clause is required with the where-clause. A by-clause is also required when the specified records are disjoint from the qualified records.

If both Collect File items and database components are specified in the retrieval-clause, the where-clause must contain one or more Collect File conditions.

Format

```
PRINT [/[options] [, TITLE comment]/] retrieval-clause

     [[,ordering-clause] where-clause] :
```

options is a list of one or more of the following format options, separated by commas. Defaults are listed in the first column.

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

For information on the format options, see Chapter 15, "Changing Output Format: Format Options."

TITLE including the TITLE option allows you to insert a comment within the output. Include the comma before the keyword TITLE only if you specify format options.

comment can be any string of characters not containing the command terminator or a slash, not beginning with a left parenthesis and not with the letters B, D, F, L, R, or S followed by a left parenthesis. The comment is displayed with the echo of the command in the Message File. See **Using the TITLE Option** on page 9-9 for more information.

Abbreviations: PRINT = PR

retrieval-clause specifies a list of one or more retrieval objects, separated by commas. See **The Retrieval-Clause** on page 9-2 for details.

ordering-clause specifies the order of selected data records. See Chapter 16, "Specifying Output Order: Ordering-Clause" for details. If you specify an ordering-clause, you must also specify a where-clause.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Considerations

- The output format for REAL and DOUBLE values is E-notation, that is ±n.nE±ee, where n.n is a mantissa giving the value and ee is an exponent giving the value's magnitude. A REAL value uses 14 characters, and a DOUBLE value uses 24 characters. REAL output consists of a sign, a single integer, a decimal point, 7 decimal positions, and an exponent, that is, ±n.nnnnnnnE±ee. DOUBLE output is similar, except that it has 17 decimal positions, ±n.nnnnnnnnnnnnnnnnE±ee. Some samples of E-notation are 1.23E+02, -1.23E10, 234.65E01.

- You can choose the output format for UNDEFINED items with the HEX OFF/HEX ON format option. You can set this option with the FORMOP command or any retrieval command, similar to specifying other format options.

  HEX OFF (the default) means display format. Any unprintable characters in the PRINT output are displayed in HEX notation (as though HEX ON were in effect). HEX ON means HEX notation format. A missing value (null) is displayed as -NULL- for either format.

Examples

Example 1

This PRINT command specifies two items, C1 (EMPLOYEE NUMBER) and C2 (LAST NAME). The values are listed in logical order; that is, they reflect the order in which the C0 records were entered into the database.

```
PRINT C1, C2:

1*  1043
2* Gibson
1*  1120
2* Reid
1*  1265
2* Slye
1*  1001
2* Waterhouse
1*  1003
2* SALAZAR
1*  1002
2* BOWMAN
1*  1101
2* NATHANIEL
   .
   .
   .
```

Example 2

This PRINT command contains a where-clause, and the retrieval-clause specifies a record, C100 (POSITION WITHIN COMPANY). Because the TREE format option is in effect, descendant data records are also displayed.

```
PRINT C100 WHERE C1 EQ 1265:

101* GENERAL MAINTENANCE
102* ADMINISTRATION & FINANCE
103* SQT
104* SUPPORT
105* 04/02/1991
  111*       $5.25
  112* HOURLY
  113* 04/02/1991
  114*      $55.73
    121* 05/31/1991
    122*   80.00
    123*   12.00
    124*     $514.50
    125*      $37.20
    126*     $458.77
    121* 04/30/1991
    122*   80.00
    124*     $420.00
    125*      $30.24
    126*     $364.27
```

Example 3

This PRINT command contains a where-clause, and the retrieval-clause specifies two items, C1 and C2, and a record, JOB SKILLS.

```
PRINT C1, C2, JOB SKILLS WHERE C1 EQ 1120:

1*  1120
2* REID

  201* PL/1
  202* EXCELLENT
  203*   5
1*  1120
2* REID

  201* COBOL
  202* GOOD
  203*   1
1*  1120
2* REID

  201* FORTRAN
  202* GOOD
  203*   1
```

Chapter 13

Retrieving Key Value Information: TALLY

Format 13-2
Considerations 13-2
Examples 13-2

The TALLY command provides summary information for key item values. The TALLY command

- allows you to request tallies of several key items with one command
- provides a statistical summary for each key item tallied.

The TALLY command has two options: EACH and ALL. EACH (the default) is set at the beginning of each session. If you include either option in a TALLY command, that option remains in effect until you either issue a TALLY command specifying the other option or exit the software.

The EACH option lists each distinct value stored in the database for the specified item(s), as well as the number of times each value occurs. The ALL option lists the minimum and maximum values stored for the requested key item(s). In both cases, the values are displayed in collating sequence. (See Appendix B.)

Regardless of the option in effect, the TALLY command displays the component name of the item being tallied, the count of distinct values stored for the item, and the count of all values (excluding nulls) stored for the item. For example, if the values for component GENDER are MALE and FEMALE with 500 occurrences of MALE and 300 occurrences of FEMALE, the count of distinct values is 2, and the count of all stored values is 800.

In all cases, CHARACTER, TEXT, and UNDEFINED items are displayed as left-justified, and numeric items and dates are displayed as right-justified.

A single TALLY command can include up to 96 items. When multiple items are included in one command, the TALLY output for each key item is displayed one below the other in the order they were entered in the command.

The values for REAL and DOUBLE items use 14 and 24 characters, respectively.

With the HEX OFF format option in effect (the default), TALLY output is in display format for UNDEFINED item values; unprintable characters are displayed as periods and message -304- appears. HEX ON means HEX notation format. The column heading is HEX VALUE.

Format

```
TALLY |[/EACH/]  items  :
      |[/ALL/]
```

items is a list of one or more key item names or component numbers, separated by commas. The maximum is 96 items.

Considerations

- The SINGLE/DOUBLE SPACE format option affects the TALLY output. That is, the output from a TALLY command is either single-spaced or double-spaced, depending on the format option in effect.
- The minimum and maximum specifications of a LIMIT command in effect when a TALLY command is issued affect the TALLY command output. Only those values occurring within the specifications are included in the output. In addition, summary information is displayed only for those values that qualify according to the LIMIT specifications.
- Only items for which you have R-authority can be specified in a TALLY command.
- With Security by Entry, only the master password holder can issue a TALLY command.

Examples

Example 1

This example shows TALLY command output with the EACH option in effect.

```
TALLY/EACH/ C8:

****************************************
 ITEM-        ETHNIC ORIGIN
****************************************
OCCURRENCES   VALUE
----------------------------------------
        2     AMERICAN INDIAN
        2     ASIAN
        7     BLACK
       31     CAUCASIAN
        8     HISPANIC
----------------------------------------
        5 DISTINCT VALUES
----------------------------------------
       50 TOTAL OCCURRENCES
----------------------------------------
```

Example 2

This example shows TALLY command output with the ALL option in effect.

```
TALLY/ALL/ C8;

*****************************************
 ITEM-        ETHNIC ORIGIN
*****************************************
 MINIMUM-     AMERICAN INDIAN
-----------------------------------------
 MAXIMUM-     HISPANIC
-----------------------------------------
        5 DISTINCT VALUES
-----------------------------------------
       50 TOTAL OCCURRENCES
-----------------------------------------
```

Example 3

This example shows TALLY command output with the EACH option and a LIMIT command in effect.

```
LIMIT 2, 5:

TALLY/EACH/C8:

*****************************************
 ITEM-        ETHNIC ORIGIN
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
        2     AMERICAN INDIAN
        2     ASIAN
-----------------------------------------
        2 DISTINCT VALUES
-----------------------------------------
        4 TOTAL OCCURRENCES
-----------------------------------------
```

Example 4

This example shows a TALLY command specifying several key items. Assuming this is the first TALLY command issued after you enter the software, the EACH option is in effect.

```
TALLY C7, C8, C9, C102:

*****************************************
 ITEM-        GENDER
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
       16     FEMALE
       34     MALE
-----------------------------------------
        2 DISTINCT VALUES
-----------------------------------------
       50 TOTAL OCCURRENCES
-----------------------------------------
*****************************************
 ITEM-        ETHNIC ORIGIN
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
        2     AMERICAN INDIAN
        2     ASIAN
        7     BLACK
       31     CAUCASIAN
        8     HISPANIC
-----------------------------------------
        5 DISTINCT VALUES
-----------------------------------------
       50 TOTAL OCCURRENCES
-----------------------------------------
*****************************************
 ITEM-        EMPLOYEE STATUS
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
       49     FULL TIME
        1     HALF TIME
-----------------------------------------
        2 DISTINCT VALUES
-----------------------------------------
       50 TOTAL OCCURRENCES
-----------------------------------------
*****************************************
 ITEM-        DEPARTMENT
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
        8     ADMINISTRATION & FINANCE
       11     EXECUTIVE
       18     INFORMATION SYSTEMS
       21     MARKETING
-----------------------------------------
        4 DISTINCT VALUES
-----------------------------------------
       58 TOTAL OCCURRENCES
-----------------------------------------
```

Chapter 14

Retrieving Value Streams: UNLOAD

Format 14-2
Unloading a Collect File 14-3
Considerations 14-4
Examples 14-5

The UNLOAD command retrieves data in a format suitable for use in the Data File. With the UNLOAD command, you can

- produce output in a value stream format for subsequent use with the LOAD command
- produce output in a format suitable for use with the *DATA* system string in the QUEUE language
- display values of ad hoc, stored, and system functions
- include a where-clause (optional)
- produce output in logical order when no where-clause is specified
- control output order when a where-clause is specified
- change format options
- unload an entire Collect File.

The UNLOAD command produces output in consecutive 80-character lines. At the end of the output, the software places a system separator (the asterisk, by default) to mark the end of the data.

When the retrieval-clause does not specify records, except in by-phrases or with the COUNT system function, the output is a series of values separated by system separators.

When the retrieval-clause includes only records and the TREE format option is in effect, the output is a series of data trees whose top records are occurrences of the specified records. The data trees are displayed in value stream format. Each data tree starts a new line, and the data for the tree end with an entry terminator, for example, END*. If the RECORD format option is in effect, only data records that are occurrences of the specified records are retrieved. In this case, each data record ends with an entry terminator.

When you issue an UNLOAD command without format options, a retrieval-clause, or a where-clause (that is, UNLOAD:), the output consists of only a system separator in column 1, thereby signaling end-of-data on the Report File. You can use this format, for example, to mark the separation between multiple blocks of PRINT or LIST output.

Like all retrieval commands, UNLOAD produces output that is written to the Report File. Since the main purpose of the UNLOAD command is to retrieve data that can later be used as input, you should specify a Report File other than the printer before issuing an UNLOAD command. Do this with the REPORT FILE IS command, which is described in **Changing Report Files: REPORT FILE IS** on page 38-17.

The file you specify in the REPORT FILE IS command can then become the Data File used with the LOAD command or with the QUEUE language *DATA* system string. You can also use the file as input to an application program.

Because UNLOAD places output in columns 1 through 80, uncontrollable line and page spacing is likely when the JCL DD statement declaring the output file specifies that column 1 is to be interpreted as a carriage control character and not as data. In other words, this can happen if the DD statement specifies a printer and the RECFM subparameter includes the character A. Also, the 80-character output requires an LRECL subparameter of 80.

Format

```
UNLOAD :

UNLOAD [/[options] [, TITLE comment]/] retrieval-clause
     [[,ordering-clause] where-clause] :

UNLOAD/COLLECT FILE [,options] [, TITLE comment]/ :
```

options is a list of one or more of the following format options, separated by commas. Defaults are listed in the first column.

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

See Chapter 15, "Changing Output Format: Format Options" for details on format options.

Abbreviations:
COLLECT FILE = CF
UNLOAD = UN

TITLE	including the TITLE option allows you to insert a comment within the output. Include a comma before the keyword TITLE only if you specify format options.
comment	can be any string of characters not containing the command terminator or a slash, not beginning with a left parenthesis, and not starting with the letters B, D, F, L, R, or S followed by a left parenthesis. The comment is displayed with the echo of the command in the Message File. See **Using the TITLE Option** on page 9-9 for more information.
retrieval-clause	specifies a list of one or more retrieval objects, separated by commas. See **The Retrieval-Clause** on page 9-2 for details.
ordering-clause	specifies the order of selected data records. See Chapter 16, "Specifying Output Order: Ordering-Clause" for details. If you specify an ordering-clause, you must also specify a where-clause.
where-clause	selects specific data records. See Chapter 24, "The Where-Clause" for details.

Unloading a Collect File

When unloading a Collect File, the entire Collect File is unloaded in value stream format as though it were a database with all items at level 0. Component numbers are 1 to *n*, where *n* is the number of items in the Collect File. Along with letting you describe the Collect File in DEFINE format, this command allows you to create a new database from a Collect File.

When you issue an UNLOAD command for a Collect File, the software unloads the entire file. You cannot use a retrieval-clause, ordering-clause, or where-clause. You can specify format options, but they have little meaning for UNLOAD/CF/. (The CF option is not global like the other format options; it does not stay in effect from command to command.) The Collect File uses scratch pads and is not a user file that can be accessed after the session. Therefore, if the Collect File is to be unloaded, it must be done within the session that collected the data.

Considerations

- When the retrieval-clause of an UNLOAD command specifies disjoint records (or items in disjoint records) and a where-clause is used as well, the retrieval-clause must contain a by-clause. See Chapter 17, "Displaying Values Together: By-Clause."

- The NULL format option does not affect UNLOAD output. If a data item is null, neither the value nor the item number is unloaded.

- If the database or Collect File exists but there are no records, the output is a single asterisk.

- The ECHO/*string*/ command, which is a system-wide command, is useful for creating commands to read the output of UNLOAD commands. For information on the ECHO command, see Chapter 43, "Echoing Commands: ECHO."

- If the current system separator occurs in a CHARACTER, TEXT, or UNDEFINED value being unloaded, message -290- appears.

  ```
  -290- DATA VALUE CONTAINS CURRENT SYSTEM SEPARATOR
  ```

  This message tells you that one or more unloaded values contain the current system separator, which could cause problems later when the software tries to read the unloaded Data File.

  The OPT000 execution parameter allows you to turn off the check for the system separator. It also allows you to specify that you want the software to check for all characters that could be used as the system separator.

- The output format for REAL and DOUBLE values is E-notation, that is ±n.nE±ee, where n.n is a mantissa giving the value and ee is an exponent giving the value's magnitude. A REAL value uses 14 characters, and a DOUBLE value uses 24 characters. REAL output consists of a sign, a single integer, a decimal point, 7 decimal positions, and an exponent, that is, ±n.nnnnnnnE±ee. DOUBLE output is similar, except that it has 17 decimal positions, ±n.nnnnnnnnnnnnnnnnE±ee. Some samples of E-notation are 1.23E+02, -1.23E10, 234.65E01.

- You can specify the output format for UNDEFINED items with the HEX OFF/HEX ON format option. HEX OFF (the default) means display format. HEX ON means HEX notation format. You can specify this format option with a FORMOP command or any retrieval command, similar to specifying other format options. For UNLOAD output, an UNDEFINED value that contains unprintable characters appears in HEX notation (as though HEX ON were in effect). Trailing binary zeros are considered insignificant.

Examples

Example 1

The retrieval-clause in this UNLOAD command specifies two items from the ENTRY record: EMPLOYEE NUMBER and LAST NAME.

```
UNLOAD C1, C2:

 1043*GIBSON* 1120*REID* 1265*SLYE* 1001*WATERHOUSE* 1003*SALAZAR* 1002*BOWMAN*
1101*NATHANIEL* 1006*GARRETT* 1004*KNIGHT* 1007*BROWN* 1062*LITTLEJOHN* 1015*SCH
OLL* 1327*BROOKS* 1145*JUAREZ* 1050*AMEER* 1161*RICHARDSON* 1017*WAGGONNER* 1119
*GOODSON* 1313*SMITH* 1234*SHROPSHIRE* 1031*CHAN* 1011*VAN HOTTEN* 1217*RODRIGUE
Z* 1133*WILLIAMSON* 1077*GIBSON* 1074*MILLSAP* 1105*FAULKNER* 1067*CAHILL* 1009*
JONES* 1247*GARCIA* 1266*REDFOX* 1008*HERNANDEZ* 1123*FREEMAN* 1049*FERNANDEZ* 1
092*POLANSKI* 1010*SMITH* 1213*JOHNSON* 1012*QUINTERO* 1083*SAVAGE* 1139*SEATON*
 1071*CANADY* 1156*REED* 1005*KNAPP* 1024*MUELLER* 1086*SCHMIDT* 1238*THROCKMORT
ON* 1261*COLLINS* 1219*KAATZ* 1112*JONES* 1272*SMITH**
```

Example 2

This UNLOAD command specifies the record C100 (POSITION WITHIN COMPANY) in the retrieval-clause and includes a where-clause. Notice that descendant records are also unloaded because the TREE option is in effect by default.

```
UNLOAD C100 WHERE C1 EQ 1265:

101*GENERAL MAINTENANCE*102*ADMINISTRATION & FINANCE*103*SQT*104*SUPPORT*105*04/
02/1991*110*111*$5.25*112*HOURLY*113*04/02/1991*114*$55.73*120*121*05/31/1991*12
2* 80.00*123* 12.00*124*$514.50*125*$37.20*126*$458.77*120*121*04/30/1991*122* 8
0.00*124*$420.00*125*$30.24*126*$364.27*END*
 *
```

Example 3

This UNLOAD command specifies the items EMPLOYEE NUMBER and LAST NAME, the JOB SKILLS record, and a where-clause.

```
UN EMPLOYEE NUMBER, LAST NAME, JOB SKILLS WH EMPLOYEE NUMBER EQ 1120:

1120*REID*201*PL/1*202*EXCELLENT*203* 5*END*
 1120*REID*201*COBOL*202*GOOD*203* 1*END*
 1120*REID*201*FORTRAN*202*GOOD*203* 1*END*
*
```

Example 4

This UNLOAD command is the same as in Example 2; however, because the RECORD format option is specified, only values from the C100 selected data record are retrieved. That is, data from descendant records are not retrieved.

```
UNLOAD/RECORD/C100 WHERE C1 EQ 1265:

101*GENERAL MAINTENANCE*102*ADMINISTRATION & FINANCE*103*SQT*104*SUPPORT*105*
*79/04/02/1991*END*
```

Example 5

This UNLOAD command includes items, system functions, and ad hoc functions in the retrieval-clause. Notice that because the software processes system functions first, the results of SUM C126 appear first in the UNLOAD output. Also, component numbers do not appear when you are unloading output for specific items. Component numbers are included only when you specify a schema record (or ENTRY for entire logical entries). (A PRINT command that retrieves the same data and its resulting output are included for comparison.)

```
UN C1, C2, SUM C126, (C124 - C114) WH C1 EQ 1265:

$823.04* 1265*SLYE*$458.77* 1265*SLYE*$364.27**



PR C1, C2, SUM C126, (C124-C114) WH C1 EQ 1265:

SUM 126*  $823.04
1*  1265
2* SLYE
10001*  $458.77
1*  1265
2* SLYE
10001*  $364.27
```

Chapter 15

Changing Output Format: Format Options

FORMAT OPTIONS 15-2
- *Considerations for Changing Format Options 15-2*
- *Controlling Indention in Reports: INDENT/BLOCK 15-4*
- *Controlling Spacing in Reports: SINGLE SPACE/DOUBLE SPACE 15-5*
- *Controlling Format of Item Labels: NUMBER/NAME 15-10*
- *Controlling Display of Item Labels: STUB/STUB SUPPRESS 15-12*
- *Controlling Display of Nulls: NULL SUPPRESS/NULL 15-14*
- *Controlling Treatment of Nulls in Functions: ZERO SUPPRESS/ZERO 15-15*
- *Controlling Data Tree Output: TREE/RECORD 15-17*
- *Controlling Repetition of Output: REPEAT/REPEAT SUPPRESS 15-18*
- *Controlling UNDEFINED Output Format: HEX OFF/HEX ON 15-21*

THE FORMOP COMMAND 15-22
- *Format 15-22*
- *Examples 15-23*

FORMAT OPTIONS

You can modify the format of output by specifying alternate format options. There are eight pairs of format options. One in each pair is the default that is in effect at the start of each session. An option remains in effect until you either specify the opposite option (even if you change processors) or exit the software.

You can change format options by issuing the following QUEST commands:

- COLLECT
- FORMOP
- LIST
- PRINT
- UNLOAD

Illus. 15.1 on page 15-3 shows the format options available and the commands affected by each option. (A legend for the illustration is shown below.)

LEGEND:	
A	Indicates that the option affects the command.
AF	Applies to all lines. Blank lines from double-spacing are included in the number of lines specified for a page by the F(*lines-per-page*) specification in a LIST command.
AD	Affects default column headings only in a LIST command.
AS	Applies only when STUB is in effect.
AP	Applies only to an arithmetic expression in parentheses.
AR	Affects MAP commands that cause restructuring. If RECORD is in effect, only level 0 records are kept.
AW	Affects only the results of LIST, PRINT, UNLOAD, and COLLECT commands with a where-clause.

Considerations for Changing Format Options

- If conflicting format options are specified in one command, the one given last will be in effect.

- As the software scans a list of format options, it makes each one effective immediately. Therefore, if an error occurs in a format option list, all options before the error are in effect.

Illus. 15.1 Effects of Format Options

QUEST Retrieval Commands

	DESCRIBE	LIST	PRINT	COLLECT	RELOAD	TALLY	UNLOAD	UNLOAD/CF/
SINGLE SPACE DOUBLE SPACE	A	AF	A	-	-	A	A	A
INDENT BLOCK	-	-	A	-	-	-	-	-
STUB STUB SUPPRESS	-	AD	A	-	-	-	-	-
NUMBER NAME	-	-	AS	-	-	-	-	-
NULL SUPPRESS NULL	-	-	A	-	-	-	-	-
ZERO SUPPRESS ZERO	-	AP	AP	AP	-	-	AP	-
TREE RECORD	-	-	A	-	A	-	A	-
REPEAT SUPPRESS	-	AW	AW	AW	-	-	AW	-
HEX OFF HEX ON	_	A	A	A	-	A	A	

QUEST Update Commands

	ADD	CHANGE	ASSIGN
ZERO SUPPRESS ZERO	AP	AP	AP

Other Processor Commands

	MAP (DEFINE)	DECLARE (REPORT)	GENERATE (REPORT)
SINGLE SPACE DOUBLE SPACE	-	-	A
ZERO SUPPRESS ZERO	-	AP	-
TREE RECORD	AR	-	-

Controlling Indention in Reports: INDENT/BLOCK

The INDENT/BLOCK format options control the indention of lower level items written to the Report File. INDENT is the default option and is cancelled when you specify the BLOCK option.

The INDENT/BLOCK format option affects only PRINT command output.

When you issue a PRINT command to display items from more than one level, hierarchical level differences are indicated by a two-space indention between levels. This indention occurs even if STUB SUPPRESS is in effect and with the NAME/NUMBER option.

When you want the lower level items to be blocked and left-justified, specify the BLOCK option.

Example 1

This PRINT command was issued with the INDENT option in effect. Notice the indention of item C102 at level 1 and item C112 at level 2.

```
PRINT C2, C3, C102, C112 WHERE C1 EQ 1265:

2* SLYE
3* LEONARD R.
  102* ADMINISTRATION & FINANCE
    112* HOURLY
```

Example 2

This PRINT command specifies the BLOCK option. With BLOCK in effect, there is no indention to indicate items from different levels.

```
PRINT /BLOCK/ C2, C3, C102, C112 WHERE C1 = 1265:

2* SLYE
3* LEONARD R.
102* ADMINISTRATION & FINANCE
112* HOURLY
```

Note: the same results would occur if you were to issue the following two separate commands:

```
LIST /BLOCK/:
PRINT C2, C3, C102, C112 WHERE C1 = 1265:
```

Controlling Spacing in Reports: SINGLE SPACE/DOUBLE SPACE

The SINGLE SPACE/DOUBLE SPACE format options control vertical spacing of output to the Report File. SINGLE SPACE is the default option and is cancelled by specifying DOUBLE SPACE.

This option affects the output from DESCRIBE, LIST, PRINT, TALLY, and UNLOAD commands.

When SINGLE SPACE is in effect, output appears in successive lines. The only exceptions are as follows:

- A blank line precedes each data record in PRINT command output when the retrieval-clause specifies a record.
- A blank line appears in PRINT command output for empty data records when their parent and children records are displayed or when the empty records are occurrences of the record specified in the retrieval-clause. The blank line is displayed only if NULL SUPPRESS is in effect. A blank line is also displayed for records with no schema items. In these cases, three lines appear between data records.
- A blank line follows the date displayed after the title line, and a blank line precedes the footing line of LIST command output.

When DOUBLE SPACE is in effect, the software inserts a blank line after each line sent to the Report File, including after the blank lines mentioned in the exceptions above.

When used with the LIST command, double spacing applies to all lines printed, including column headings. Blank lines from double spacing are included in the count of the number of lines specified for a page by F(*lines-per-page*).

If double spacing is in effect for UNLOAD, the file produced causes errors during the scan in a loading process because of unwanted blanks between lines.

Example 1

This example shows a PRINT command with the SINGLE SPACE and NULL SUPPRESS options in effect and the NAME option specified. (When the NULL option is in effect, null items are displayed as -NULL-, as shown on the next page.)

```
PRINT /NAME/ C0 WHERE C2 EQ GARRETT:

  EMPLOYEE NUMBER*  1006
  LAST NAME* GARRETT
     .
     .
     .
    POSITION TITLE* VICE-PRESIDENT MARKETING
    DEPARTMENT* EXECUTIVE
    MANAGER* HEB
    POSITION TYPE* CORPORATE OFFICER
    START DATE* 01/01/1991

      PAY RATE*   $4,500.00
      PAY SCHEDULE* MONTHLY/BONUS
      EFFECTIVE DATE* 01/01/1991
      CURRENT DEDUCTION*     $711.10

        PAYROLL MONTH* 05/31/1991
        REGULAR HOURS*  184.00
        GROSS PAY*   $4,500.00
        FEDERAL TAX DEDUCTION*     $527.20
        NET PAY*   $3,788.90

           .
           .
           .
    SKILL TYPE* FINANCIAL AUDITING
    PROFICIENCY* EXCELLENT
    YEARS OF EXPERIENCE*   7

       .
       .
       .
    INTEREST* SOFTBALL
    AFFILIATION* AUSTIN SOFTBALL LEAGUE

    INTEREST* PINGPONG

    INTEREST* CHESS
    AFFILIATION* UNIVERSITY CHESS CLUB

    INTEREST* SCIENCE FICTION


      SCHOOL* TEXAS CHRISTIAN UNIVERSITY
      DEGREE/CERTIFICATE* BS
      DATE COMPLETED* 08/01/1968
      MAJOR FIELD* HISTORY


      SCHOOL* UNIVERSITY OF WASHINGTON
      DEGREE/CERTIFICATE* MS
      DATE COMPLETED* 05/30/1973
      MAJOR FIELD* EDUCATION
      MINOR FIELD* HISTORY
```

Example 1 (continued)

```
PRINT /NULL/ C0 WHERE C2 EQ GARRETT:

  EMPLOYEE NUMBER*  1006
  LAST NAME* GARRETT
     .
     .
     .
    POSITION TITLE* VICE-PRESIDENT MARKETING
    DEPARTMENT* EXECUTIVE
    MANAGER* HEB
    POSITION TYPE* CORPORATE OFFICER
    START DATE* 01/01/1991
    END DATE* -NULL-

      PAY RATE*   $4,500.00
      PAY SCHEDULE* MONTHLY/BONUS
      EFFECTIVE DATE* 01/01/1991
      CURRENT DEDUCTION*     $711.10

        PAYROLL MONTH* 05/31/1991
        REGULAR HOURS*  184.00
        OVERTIME HOURS* -NULL-
        GROSS PAY*   $4,500.00
        FEDERAL TAX DEDUCTION*     $527.20
        NET PAY*   $3,788.90

           .
           .
           .
    SKILL TYPE* FINANCIAL AUDITING
    PROFICIENCY* EXCELLENT
    YEARS OF EXPERIENCE*   7

       .
       .
       .
    INTEREST* SOFTBALL
    AFFILIATION* AUSTIN SOFTBALL LEAGUE
    COMMENT* -NULL-

    INTEREST* PINGPONG
    AFFILIATION* -NULL-
    COMMENT* -NULL-

    INTEREST* CHESS
    AFFILIATION* UNIVERSITY CHESS CLUB
    COMMENT* -NULL-

    INTEREST* SCIENCE FICTION
    AFFILIATION* -NULL-
    COMMENT* -NULL-

      SCHOOL* TEXAS CHRISTIAN UNIVERSITY
      DEGREE/CERTIFICATE* BS
      DATE COMPLETED* 08/01/1968
      MAJOR FIELD* HISTORY
      MINOR FIELD* -NULL-

      SCHOOL* UNIVERSITY OF WASHINGTON
      DEGREE/CERTIFICATE* MS
      DATE COMPLETED* 05/30/1973
      MAJOR FIELD* EDUCATION
      MINOR FIELD* HISTORY
```

Example 2

This example shows the effect of DOUBLE SPACE in a LIST command. When SINGLE SPACE is in effect, as in the first output, the report fits on one page. Notice that 30 is specified in the paging specification F(30).

```
LIST /TITLE D(20)SKILL TYPE REPORT, R(10)LAST  NAME,
   L(16)   SKILL TYPE,R(11)PROFICIENCY,R(6)  YEAR+    OF+EXPERI,
   B(8)S++ENCE,
   F(30)THIS IS A ONE-PAGE REPORT IF SINGLE SPACE IS IN EFFECT/
   C2, C201, C202, C203 WHERE C1 EQ 1001:

                SKILL TYPE REPORT                          1
                   02/19/1990

*  LAST NAME    SKILL TYPE         PROFICIENCY     YEARS
                                                     OF
                                                EXPERIENCE
***
* WATERHOUSE   PUBLIC RELATIONS   EXCELLENT          19
* WATERHOUSE   FRENCH             EXCELLENT          21
* WATERHOUSE   COBOL              GOOD               10
* WATERHOUSE   SYSTEMS ANALYSIS   EXCELLENT          21
               & DESIGN
* WATERHOUSE   PUBLIC SPEAKING    EXCELLENT          12
* WATERHOUSE   ASSEMBLER          FAIR                4
* WATERHOUSE   ACCOUNTING         EXCELLENT          12

THIS IS A ONE-PAGE REPORT IF SINGLE SPACE IS IN EFFECT
```

(continued next page)

Example 2 (continued)

However, the report needs two pages when DOUBLE SPACE is in effect.

```
                    SKILL TYPE REPORT                              1
                       02/19/1990

*  LAST NAME     SKILL TYPE          PROFICIENCY      YEARS
                                                        OF
                                                   EXPERIENCE
***
* WATERHOUSE    PUBLIC RELATIONS    EXCELLENT          19
* WATERHOUSE    FRENCH              EXCELLENT          21
* WATERHOUSE    COBOL               GOOD               10
* WATERHOUSE    SYSTEMS ANALYSIS    EXCELLENT          21
                & DESIGN
* WATERHOUSE    PUBLIC SPEAKING     EXCELLENT          12

THIS IS A ONE-PAGE REPORT IF SINGLE SPACE IS IN EFFECT

                    SKILL TYPE REPORT                              2
                       02/19/1990

*  LAST NAME     SKILL TYPE          PROFICIENCY      YEARS
                                                        OF
                                                   EXPERIENCE
***
* WATERHOUSE    ASSEMBLER           FAIR               4
* WATERHOUSE    ACCOUNTING          EXCELLENT          12

THIS IS A ONE-PAGE REPORT IF SINGLE SPACE IS IN EFFECT
```

Controlling Format of Item Labels: NUMBER/NAME

The NUMBER/NAME format options specify whether the item number or the item name is to be displayed in output written to the Report File. NUMBER is the default option and is cancelled by specifying NAME.

The NUMBER/NAME format option affects only PRINT command output when the STUB option is in effect.

When NUMBER is in effect, the item number and the system separator precede each value written to the Report File by the PRINT command.

If NAME is in effect, the item name appears instead of the item number. The item name is followed by the system separator.

Specifying STUB SUPPRESS does not cancel the NUMBER/NAME option; however, it prevents the NUMBER/NAME option from affecting the output. For example, if STUB SUPPRESS is specified when NAME is in effect, PRINT command output does not contain item names and system separators. These are shown only after STUB is specified to cancel STUB SUPPRESS.

The NUMBER/NAME option also affects output of stored functions and system functions. When NUMBER is in effect, the component numbers of stored functions are displayed, and item numbers follow the names of the system functions. When NAME is in effect, stored function names are displayed, and item names follow the names of the system functions. The results of ad hoc functions are always preceded by a software-assigned number and a system separator, unless STUB SUPPRESS is in effect.

Collect File item names are always used regardless of the setting of the NAME/NUMBER format option, because there are no numbers explicitly associated with Collect File items.

Example 1

This example shows a PRINT command and resulting output when NUMBER (the default option) is in effect. 10001 is the label assigned by the software to the ad hoc function (C126/2). The stored function is as follows:

```
2002* TAX PERCENT OF PAY (DECIMAL FUNCTION (((C125/C124) * 100) ))


PRINT SUM C126, C2, C126,(C126/2), *C2002* WH C1 EQ 1265 AND C125 GT 35.00:

SUM 126* $458.77
2* SLYE
      126*     $458.77
10001*  229.385
2002*   7.230
```

Abbreviations: NUMBER = NUMB, NUM

Example 2

This example shows the same PRINT command and resulting output when NAME is in effect.

```
PR /NAME/ SUM C126, C2, C126, (C126/2),*C2002* WH C1 EQ 1265 AND C125
   GT 35.00:

SUM NET PAY* $458.77:
LAST NAME* SLYE
      NET PAY*     $458.77
10001*  229.385
TAX PERCENT OF PAY*   7.230
```

Example 3

In this command, STUB SUPPRESS overrides the NAME option, although it does not cancel it. That is, if the STUB option is specified in a subsequent command, component names will be displayed.

```
PRINT /NAME, BLOCK, STUB SUPPRESS/ C0 WH C201 EXISTS:

 1043
GIBSON
MOLLY I.
10/01/1990
11/13/1961
462-01-0234
FEMALE
BLACK
FULL TIME
323 XT227
  40.00
  56.00
 106
2311 HANSFORD
AUSTIN TX
78753
TECHNICAL WRITER
INFORMATION SYSTEMS
JC
PROFESSIONAL
10/01/1990
  $2,100.00
MONTHLY
10/01/1990
    $455.65
  1.0
HUSBAND EMPLOYED AS INSTRUCTOR
  2.0
IN THE MARKETING DEPARTMENT
  3.0
NAME: GEORGE J. GIBSON
OPERATIONS
INFORMATION SY
JC
PROFESSIONAL
    .
    .
    .
```

Controlling Display of Item Labels: STUB/STUB SUPPRESS

The STUB/STUB SUPPRESS format options control the display of item labels in the Report File. STUB is the default option and is cancelled when you specify STUB SUPPRESS.

The STUB/STUB SUPPRESS format option is affected by the NUMBER/NAME format option. When the STUB option is in effect, each value written to the Report File by the PRINT command is preceded by the item number (if the NUMBER option is in effect) or by the item name (if the NAME option is in effect). Collect File items have names only, not item numbers. The item number (or name) is followed by the system separator and one blank space.

The STUB SUPPRESS option eliminates the item name (or number) and the system separator in PRINT command output. It also eliminates default column headings in LIST command output.

The STUB/STUB SUPPRESS format option also affects the output of stored functions, ad hoc functions, and system functions. If STUB SUPPRESS is in effect, the results of the computations appear without labels.

The STUB option must be in effect for the NUMBER/NAME option to affect output. Thus, if the STUB SUPPRESS option is in effect, item numbers or names do not appear.

Abbreviations: SUPPRESS = SUP

Example 1

This example shows a PRINT command and resulting output when STUB (the default option) and NAME are in effect. (10001 is the label assigned by the software to the ad hoc function (C126/2).) The stored function is as follows:

```
2002* TAX PERCENT OF PAY (DECIMAL FUNCTION (((C125/C124) * 100) ))


PR SUM C126, C2, C126, (C126/2), *C2002* WH C1 EQ 1265 AND C125 GT 35.00:

SUM NET PAY* $458.77
LAST NAME* SLYE
      NET PAY*      $458.77
10001*  229.385
TAX PERCENT OF PAY*   7.230
```

Example 2

This example shows the same PRINT command and resulting output when STUB SUPPRESS is in effect. Notice that the name of the system function (SUM) is not suppressed.

```
PR /STUB SUPPRESS/:

PR SUM C126, C2, C126, (C126/2), *C2002* WH C1 EQ 1265 AND C125 GT 35.00:

SUM $458.77
SLYE
          $458.77
 229.385
  7.230
```

Controlling Display of Nulls: NULL SUPPRESS/NULL

The NULL SUPPRESS/NULL format options control the display of nulls. When NULL SUPPRESS is in effect, the PRINT command does not display any output for null items. When NULL is in effect, the word -NULL- is displayed for those items with no assigned value. NULL SUPPRESS is the default option and is cancelled by specifying NULL.

The NULL SUPPRESS/NULL format options affect only PRINT command output. They have no effect on UNLOAD, which always suppresses null output. Neither do they affect LIST, which inserts blanks for nulls, but omits lines consisting entirely of nulls.

The NULL SUPPRESS/NULL options also affect output generated by stored functions, ad hoc functions, and system functions (except COUNT). In some cases, no output is shown as a result of some computations involving null items; that is, neither a function label nor a value is displayed. However, if NULL is in effect, the function label is shown and -NULL- appears as the function value. When the system function COUNT is used to count the number of times an item occurs, and the item specified does not occur, the result is zero, even if NULL is in effect.

The NULL SUPPRESS/NULL format options have no effect on data being written to a Collect File.

Example 1

This example shows how you can use the NULL format option to avoid possible misinterpretations when scanning output. First, the output of a PRINT command is shown when NULL SUPPRESS (the default) and NAME are in effect. The output shows, falsely, that the database contains only one entry where an employee has the title of programmer. It seems that Reid has two records for department and manager.

```
PRINT C2, C102, C103 WH C101 EQ PROGRAMMER:

2* REID
  102* INFORMATION SYSTEMS
  103* MYJ
  102* INFORMATION SYSTEMS
  103* MYJ
```

Next, the same command specifies the NULL format option. Notice now that the database actually contains two entries where the position title is programmer, but one entry contains a null for C2 (LAST NAME).

```
PRINT /NULL/ C2, C102, C103 WH C101 EQ PROGRAMMER:

2* REID
  102* INFORMATION SYSTEMS
  103* MYJ
2* -NULL-
  102* INFORMATION SYSTEMS
  103* MYJ
```

Controlling Treatment of Nulls in Functions: ZERO SUPPRESS/ZERO

The ZERO SUPPRESS/ZERO format options cause nulls to be treated as zeros in arithmetic expressions. ZERO SUPPRESS is the default and and remains in effect until ZERO is specified.

ZERO SUPPRESS/ZERO affects only numeric items and only when they are used in ad hoc functions or stored functions. When ZERO SUPPRESS is in effect, a null value produces null results for a numeric item included in an ad hoc function or stored function. When ZERO is in effect, nulls are changed to zero for items included in computations. For dates, a null value causes a null result regardless of the ZERO SUPPRESS/ZERO setting; that is, nulls for dates are not converted to zero. The occurrence of one null date causes the result to be null.

This option applies to system functions (AVG, COUNT, MAX, MIN, SIGMA, and SUM) only if the object of the system function is presented as an ad hoc function; that is, the object is contained in parentheses.

This option affects the results of updates when ad hoc functions or stored functions appear in the action-clause of update commands.

Illus. 15.2 on page 15-16 and Illus. 15.3 on page 15-16 show how the output of commands is affected by setting different combinations of these two options. In addition, the calculations used by the software in arriving at these values are shown. The database used contains only the three entries shown below, and all three items are INTEGER types.

```
101* 5
102* -NULL-
103* -NULL-
```

```
101* 7
102* 3
103* -NULL-
```

```
101* -NULL-
102* -NULL-
103* -NULL-
```

The software assigns numbers, starting with 10001, to each ad hoc function in a command. Since each command in the following table contains only one ad hoc function, the number assigned to the function output in each case is 10001.

Illus. 15.2 Effects of the ZERO Format Option

Commands	ZERO and NULL	ZERO and NULL SUPPRESS	Calculations
PR AVG C101:	AVG 101* 6	AVG 101* 6	(5+7)/2
PR AVG (C101):	AVG 10001 *4	AVG 10001 *4	(5+7+0)/3
PR AVG C103:	AVG 103* -NULL-	∅	None
PR AVG (C103):	AVG 10001* 0	AVG 10001* 0	(0+0+0)/3
PR COUNT C101:	CNT 101* 2	CNT 101* 2	# of values in {5,7}
PR COUNT (C101):	CNT 10001* 3	CNT 10001* 3	# of values in {5,7,0}
PR MIN C101:	MIN 101* 5	MIN 101* 5	min of {5,7}
PR MIN (C101):	MIN 10001* 0	MIN 10001* 0	min of {5,7,0}
PR (C101+C102):	10001* 5 10001* 10 10001* 0	10001* 5 10001* 10 10001* 0	(5+0) (7+3) (0+0)
PR AVG (C101+C102):	AVG 10001* 5	AVG 10001* 5	(5+10+0)/3

Illus. 15.3 Effects of the ZERO SUPPRESS Format Option

Commands	ZERO SUPPRESS and NULL	ZERO SUPPRESS and NULL SUPPRESS	Calculations
PR AVG C101:	AVG 101* 6	AVG 101* 6	(5+7)/2
PR AVG (C101):	AVG 10001* 6	AVG 10001* 6	(5+7)/2
PR AVG C103:	AVG 103* -NULL-	∅	None
PR AVG (C103):	AVG 10001* -NULL-	∅	None
PR COUNT C101:	CNT 101* 2	CNT 101* 2	# of values in {5,7}
PR COUNT (C101):	CNT 10001* 2	CNT 10001* 2	# of values in {5,7,}
PR MIN C101:	MIN 101* 5	MIN 101* 5	min of {5,7}
PR MIN (C101):	MIN 10001* 5	MIN 10001* 5	min of {5,7}
PR (C101+C102):	10001* -NULL- 10001* 10 10001* -NULL-	∅ 10001* 10 ∅	None (7+3) None
PR AVG (C101+C102):	AVG 10001* 10	AVG 10001* 10	10/1

Controlling Data Tree Output: TREE/RECORD

The TREE/RECORD format options control the output of lower level data records when a record is specified in the retrieval-clause of a PRINT or UNLOAD command. TREE is the default option and remains in effect until RECORD is specified.

When the retrieval-clause of a PRINT or UNLOAD command specifies a record and TREE is in effect, the software displays data trees whose top records are occurrences of the specified record. That is, the output includes the data records and their descendant records.

In contrast, when RECORD is in effect, only data records that are occurrences of the specified record are displayed. Descendant records do not appear.

If RECORD is in effect when the RELOAD command is issued, only level 0 records are reloaded. If RECORD is in effect when a MAP command that causes restructuring is issued, only level 0 records are retained. Descendant records are lost.

Example 1

The TREE format option is in effect for the following command:

```
PRINT C100 WHERE C1 EQ 1327:

101* JR SALES REPRESENTATIVE
102* MARKETING
103* MAS
104* PROFESSIONAL
105* 05/01/1991

  111*   $2,250.00
  112* MONTHLY
  113* 05/01/1991
  114*     $384.15

    121* 05/31/1991
    122*  184.00
    124*   $2,250.00
    125*     $292.20
    126*   $1,865.85
```

Example 2

When the same command is issued specifying RECORD, the resulting output does not contain values for descendant records.

```
PRINT /RECORD/ C100 WHERE C1 EQ 1327:

101* JR SALES REPRESENTATIVE
102* MARKETING
103* MAS
104* PROFESSIONAL
105* 05/01/1991
```

Controlling Repetition of Output: REPEAT/REPEAT SUPPRESS

The REPEAT/REPEAT SUPPRESS format options control the repeated display of data from common ancestors of selected data records. REPEAT is the default option and is cancelled by specifying REPEAT SUPPRESS.

When the retrieval-clause of a command with a where-clause contains objects from different levels of the database, the software processes the retrieval-clause by starting at each selected data record and obtaining the Hierarchical Table addresses of the ancestor records. The retrieval-clause is then scanned from left to right, and data are retrieved in that order.

When REPEAT is in effect and when two or more selected data records have common ancestors, data from those ancestor records are displayed each time data from a selected record are displayed.

When REPEAT SUPPRESS is in effect and successive selected data records have a common ancestor, values from that ancestor record are displayed only once, and the values from each data tree appear in logical order.

To ensure effectiveness of REPEAT SUPPRESS, you should include an ordering-clause that specifies each schema record from C0 down to the level of the parent of the target record. This type of sort groups descendant data records together under their common ancestor data records.

For REPEAT SUPPRESS, if a schema item's value is being written to a Collect File, null is written if the item's value has previously been written. That is, schema items work the same way with REPEAT SUPPRESS as they do with the PRINT, LIST, and UNLOAD commands.

The output from a command without a where-clause is always in logical order; therefore, in this case, the option has no meaning, and it is not checked by the software.

REPEAT SUPPRESS causes data not to be displayed repeatedly by checking the Hierarchical Table address(es) of the ancestor(s) of previously processed selected records, not by checking values in those ancestor records.

Abbreviations: SUPPRESS = SUP

The following examples use the second and third logical entries in the EMPLOYEE database, shown in the foldout at the end of this manual.

Example 1

For this LIST command, REPEAT is in effect, and the output is in physical order. That is, the output is ordered according to the Hierarchical addresses of the selected C110 data records. The first line shows values from records in the path of record 14, the second line shows values from records in the path of record 15, and so on. Notice that the value for C7 (MALE) and C105 (START DATE) from record 12 are repeated in the second line, even though it is from the same record as the value in the first line.

```
LIST C7, C105, C111 WHERE (C1 EQ 1120 OR C1 EQ 1265)
   AND C111 LE 1800.00:

* GENDER    START DATE      PAY RATE
***
* MALE      03/01/1989     $1,800.00
* MALE      03/01/1989     $1,650.00
* MALE      02/16/1988         $5.25
* MALE      04/02/1991     $1,500.00
```

Example 2

In this LIST command, REPEAT SUPPRESS is in effect. Because the command does not include an ordering-clause, any suppression of values from common ancestor records is accidental.

In this case, the output is the same as in the previous example, except that the value for C7 (MALE) and C105 (START DATE) are not repeated in the second line because REPEAT SUPPRESS is in effect and because it comes from the same record as the value for C7 in the first line.

```
LIST /REPEAT SUPPRESS/ C7, C105, C111 WHERE (C1 EQ 1120
   OR C1 EQ 1265) AND C111 LE 1800.00:

* GENDER    START DATE      PAY RATE
***
* MALE      03/01/1989     $1,800.00
*                          $1,650.00
* MALE      02/16/1988     $1,500.00
* MALE      04/02/1991         $5.25
```

Example 3

This LIST command includes an ordering-clause and has REPEAT SUPPRESS in effect. Notice that the ordering-clause specifies C0 and C100 as sort keys. C0 and C100 are the ancestor records of the target record.

The output is in logical order. The value for C7 from record 12 is displayed only once, and the value for C105 from record 13 is displayed only once.

```
LIST /REPEAT SUPPRESS/ C7, C105, C111, ORDERED BY C0, C100
   WHERE (C1 EQ 1120 OR C1 EQ 1265) AND C111 LE 1800.00:

* GENDER   START DATE     PAY RATE
***
* MALE     03/01/1989    $1,650.00
*                        $1,800.00
*          02/16/1988    $1,500.00
* MALE     04/02/1991        $5.25
```

Example 4

This LIST command shows how a by-clause can eliminate repetition. But because the record specified in the by-phrase (C0) is above the level of the qualified data records (C110), the output includes a value ($1,900.00) that does not satisfy the condition C111 LE 1800.00.

```
LIST BY C0, C7, C105, C111 WHERE (C1 EQ 1120 OR C1 EQ 1265)
   AND C111 LE 1800.00:

* GENDER   START DATE     PAY RATE
* MALE     03/01/1989    $1,900.00

*                        $1,800.00

*                        $1,650.00

*          02/16/1988    $1,550.00

* MALE     04/02/1991        $5.25
```

Controlling UNDEFINED Output Format: HEX OFF/HEX ON

The HEX OFF and HEX ON format options control the output format of UNDEFINED item values. HEX OFF (the default) means display format. HEX ON means HEX notation format. For example, SYSTEM 2000 would be displayed as X'E2E8E2E3C5D440F2F0F0F0'. The software displays a missing value (null) as -NULL- for either format. Trailing binary zeros are considered insignificant for PRINT and UNLOAD output.

For HEX OFF, unprintable characters in the LIST and TALLY output are displayed as periods and message -304- appears. For PRINT and UNLOAD output, an UNDEFINED value that contains unprintable characters appears in HEX notation (as though HEX ON were in effect).

The following rules apply when the LIST output is being formatted:

- The LIST command fills the specified columns for an UNDEFINED item with as many characters as possible before truncating. However, if the system truncates only trailing binary zeros, it does not display the asterisk that normally indicates truncation.

- For HEX ON, the list command requires a minimum of five columns, which will accommodate the word HEX and a blank plus a minimum one-character item name in the column heading. Also, for HEX ON, an even number of hexadecimal characters are displayed. Characters beyond the 125th byte (250 hexadecimal characters) are truncated.

- For HEX OFF, the list command does not truncate any data value that fits within the specified number of columns for the UNDEFINED item.

With HEX OFF, if the UNLOAD command unloads any values that contain the current system separator or the command delimiter, warning message -290- is issued, and the results are unpredictable. The OPT000 execution parameter allows you to disable the check for the system separator or to check for occurrences of all characters that are allowed as the system separator. Checking for all valid characters takes more time. It tells you which characters do not occur in any unloaded values and therefore could be used as the current system separator during the unload processing.

With HEX ON in effect, the TALLY output is in HEX notation. The column heading is HEX VALUE.

THE FORMOP COMMAND

Besides using retrieval commands to change format options, you can issue the FORMOP command to change format options.

The format options remain set, even if you change processors,

- until they are reset in another FORMOP command or a PRINT, LIST, UNLOAD, or COLLECT command
- until the session ends.

As the software scans each format option, it sets the option. If an error occurs in the FORMOP command, any options set before the error remain in effect.

If you issue the FORMOP command with no format options, the software ignores the command.

Format

```
FORMOP /options/  :
```

options is a list of one or more of the following format options, separated by commas. Defaults are listed in the first column.

INDENT	BLOCK
NULL SUPPRESS	NULL
NUMBER	NAME
REPEAT	REPEAT SUPPRESS
SINGLE SPACE	DOUBLE SPACE
STUB	STUB SUPPRESS
TREE	RECORD
ZERO SUPPRESS	ZERO
HEX OFF	HEX ON

For information about the format options, see the previous sections in this chapter.

Abbreviations:
NUMBER = NUMB, NUM
SUPPRESS = SUP

Examples

Example 1

This FORMOP command changes two format options. The output from a LIST command reflects the new formats.

```
FORMOP /REPEAT SUPPRESS, DOUBLE SPACE/:

LIST /TITLE L(9)LAST NAME,FIRST NAME,L(40)POSITION TITLE/
   C2, C3, C101 WH C103 EQ HEB:

* LAST NAME   FIRST NAME    POSITION TITLE

***

* NATHANIEL   DARRYL        SECRETARY

* GARRETT     OLAN M.       VICE-PRESIDENT MARKETING

*                          MANAGER OF SALES

* MILLSAP     JOEL B.       VICE-PRESIDENT INFORMATION SYSTEMS

* KNAPP       PATRICE R.    VICE-PRESIDENT ADMINISTRATION & FINANCE
```

Example 2

You can specify each format option on a separate line for easier reading.

```
FORMOP /
   SINGLE SPACE,
   BLOCK,
   STUB,
   NUMBER,
   NULL,
   REPEAT SUPPRESS,
   ZERO,
   TREE
   HEX ON
   /:
```

Chapter 16

Specifying Output Order: Ordering-Clause

Format 16-3
Considerations 16-4
Examples 16-4

When you include a where-clause in a LIST, PRINT, UNLOAD, or COLLECT command, you can also include an ordering-clause to specify the output order of the retrieved data. The same is true for the RELOAD command and for the DEFINE language MAP command.

The ordering-clause allows you to specify that output be sorted

- according to the values of one major key and several minor keys
- according to values computed from ad hoc, stored, or system functions
- according to the values of key or non-key items
- according to values that are not displayed
- in ascending or descending order for any sort key.

An ordering-clause consists of the keywords ORDERED BY followed by one or more sort keys separated by commas. A sort key can be a key or non-key item, a record, a stored function, an ad hoc function, or a system function.

The order in which the sort keys are specified in the ordering-clause determines the order in which retrieval objects are sorted for display. The values used for sorting depend on the sort key specified, as shown in the illustration on the next page. Sorting on a system function without a by-phrase is meaningless, because there would be only one value for the database as a whole. You cannot specify Collect File items in an ordering-clause.

Illus. 16.1 Values Used in Ordering-Clauses

Sort Key	Values Used for Sort
Database item	Values for the item
Ad hoc function	Result of computation
Stored function	Result of computation
System function	Result of computation
Record	Physical address of record

Each sort key can begin with the optional specification HIGH or LOW; LOW is the default. These words specify that values be ordered in ascending order (LOW) or in descending order (HIGH).

Components in an ordering-clause must be in the same path as the target record in the retrieval-clause. For example, in the EMPLOYEE database, the following command is not acceptable because C101 and C201 are in separate paths:

```
PRINT C101, ORDERED BY C201 WHERE C1 EXISTS:
```

However, the following commands are acceptable because C1, ENTRY, and C101 are in the same path:

```
PRINT C1, BY ENTRY, C203, ORDERED BY C101 WHERE C1 EXISTS:

PRINT C1, C101, ORDERED BY AVG C203 BY ENTRY WHERE C1 EXISTS:
```

When all sort keys are above the level of the target record, the order in which data from records below the level of the sort keys are displayed cannot be predicted. (See Examples 3 and 4 later in this chapter.)

When a sort key is below the target record, multiple copies of retrieved data (one for each occurrence of the sort key) can be shown. The copies are displayed according to the values for the sort keys; therefore, they do not necessarily appear together.

Functions are considered to be at the level of the lowest component identified in the function, unless the function is preceded with a by-phrase, in which case the level of the function is that of the specified record.

The software sorts the selected records obtained from the list of qualified records that results from where-clause processing. Therefore, a where-clause is required when you include an ordering-clause in a PRINT, LIST, UNLOAD, COLLECT, RELOAD, or MAP command.

Format

```
, ORDERED BY  sort-keys
```

sort-keys is a list of one or more items, records, ad hoc functions, stored functions, or system functions in the following format, separated by commas. You cannot use Collect File items as sort keys.

```
|[LOW]   unit
|[HIGH]
```

LOW specifies low-to-high (ascending) order, for example, A,B,C,D or 1,2,3,4. The default is LOW.

HIGH specifies high-to-low (descending) order, for example, Z,Y,X,W or 9,8,7,6. See Appendix B for collating sequence.

unit

```
|item
|record
|stored-function
|(adhoc-function)
|system-function
```

item is a schema item name or component number.

record is a schema record name or component number.

stored-function is a stored function name or component number with optional parameters.

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions" for details.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. A by-phrase is required if you use a system function in an ordering-clause. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions" for details.

A comma must precede the keywords ORDERED BY (or the abbreviation OB).
Abbreviations: ORDERED BY = OB

Considerations

- Digits in numeric values for textual items are treated as characters for sorting purposes; therefore, those values are not sorted numerically. (See Example 11.)
- When an ordering-clause specifies a list of sort keys whose values exceed 364 characters, the software uses the first 364 characters to sort the retrieved data records. Characters beyond the 364th character are ignored, and the software issues the following message:

```
-314- ORDERING KEY TRUNCATED AT LEAST ONCE -
```

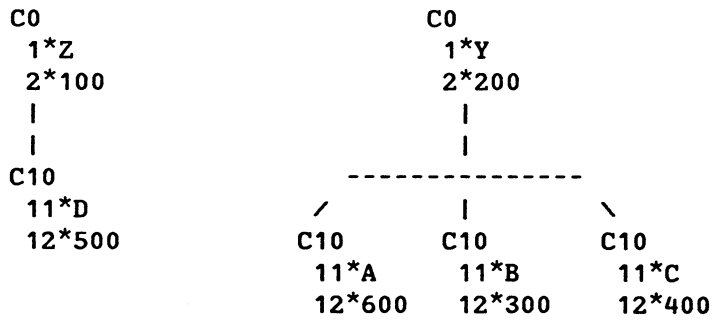
- Sort items do not have to be displayed by the action-clause.

Examples

To predict the output order from an ordering-clause and to use an ordering-clause more effectively, it is important to consider the physical addresses of the records in the Hierarchical Table.

Although these addresses are unknown to you, their relative position can be identified. (See Example 2).

To illustrate the role that Hierarchical Table addresses play in sorting output, each of the records in the database used for the following examples includes an item (either C2 or C12) identifying the address of the record, as shown below.



Example 1

Since there is no where-clause, this PRINT command produces output in logical order.

```
PRINT C1, C2, C10:

  1* Z
  2*  100

    11* D
    12*  500
  1* Y
  2*  200

    11* A
    12*  600

    11* B
    12*  300

    11* C
    12*  400
```

Example 2

Because this PRINT command includes a where-clause, the output is displayed in physical order. That is, the records are arranged numerically by Hierarchical Table address.

```
PRINT C1, C2, C10 WHERE C11 EXISTS:

  1* Y
  2*  200

    11* B
    12*  300
  1* Y
  2*  200

    11* C
    12*  400
  1* Z
  2*  100

    11* D
    12*  500
  1* Y
  2*  200

    11* A
    12*  600
```

Example 3

In this command, the ordering-clause causes the output to be ordered by the addresses of the C0 records. Therefore, the C10 record under the C0 record with address 100 precedes the C10 records that are descendants of the C10 record with address 200. The order in which the C10 records under the C0 record with address 200 are displayed cannot be predicted because the ordering-clause does not include a sort key for C10 records.

```
PRINT C1, C2, C10, ORDERED BY C0 WH C11 EXISTS:

1* Z
2*  100

  11* D
  12*  500
1* Y
2*  200

  11* C
  12*  400
1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* A
  12*  600
```

Example 4

In this command, the ordering-clause causes output to be ordered by the C1 value in the C0 records. Thus, the C10 records under the C0 record where C1 equals Y precedes the C10 record under the C0 record where C1 equals Z. The order in which the C10 records under the C0 record where C1 equals Y are displayed cannot be predicted because the ordering-clause does not include a sort key for C10 records.

```
PRINT C1, C2, C10, ORDERED BY C1 WH C11 EXISTS:

1* Y
2*  200

  11* C
  12*  400
1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* A
  12*  600
1* Z
2*  100

  11* D
  12*  500
```

Example 5

In this command, the ordering-clause causes output to be ordered by the Hierarchical Table addresses of the C10 record. Notice that the same result is obtained in Example 2, where an ordering-clause is not used. In this case, sorting by C10 is repetitious because C10 records were already in physical order after where-clause processing.

```
PRINT C1, C2, C10, OB C10 WH C11 EXISTS:

 1* Y
 2*  200

   11* B
   12*  300
 1* Y
 2*  200

   11* C
   12*  400
 1* Z
 2*  100

   11* D
   12*  500
 1* Y
 2*  200

   11* A
   12*  600
```

Example 6

In this command, the ordering-clause causes output to be ordered by the C11 value in the C10 records.

```
PRINT C1, C2, C10, OB C11 WH C11 EXISTS:

 1* Y
 2*  200

   11* A
   12*  600
 1* Y
 2*  200

   11* B
   12*  300
 1* Y
 2*  200

   11* C
   12*  400
 1* Z
 2*  100

   11* D
   12*  500
```

Example 7

In this example, the output is the same as in Example 3. However, specifying C11 as a minor sort key here guarantees that the descendant C10 records under each C0 record will be ordered by the C11 value.

```
PRINT C1, C2, C10, OB C0, C11 WH C11 EXISTS:

1* Z
2*  100

  11* D
  12*  500
1* Y
2*  200

  11* A
  12*  600
1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* C
  12*  400
```

Example 8

The output here is the same as in Example 3. However, in this case, specifying C10 as a minor sort key guarantees that the descendant C10 records under each C0 record will be displayed in physical order.

```
PRINT C1, C2, C10, OB C0, C10 WH C11 EXISTS:

1* Z
2*  100

  11* D
  12*  500
1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* C
  12*  400
1* Y
2*  200

  11* A
  12*  600
```

Example 9

The output here is the same as in Example 4. However, in this case, specifying C11 as a minor sort key guarantees that the descendant C10 records under each C0 record will be ordered by the C11 value.

```
PRINT C1, C2, C10, OB C1, C11 WH C11 EXISTS:

1* Y
2*  200

  11* A
  12*  600
1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* C
  12*  400
1* Z
2*  100

  11* D
  12*  500
```

Example 10

The output here is the same as in Example 4. However, in this case, specifying C10 as a minor sort key guarantees that the descendant C10 records under each C0 record will be displayed in physical order.

```
PRINT C1, C2, C10, OB C1, C10 WH C11 EXISTS:

1* Y
2*  200

  11* B
  12*  300
1* Y
2*  200

  11* C
  12*  400
1* Y
2*  200

  11* A
  12*  600
1* Z
2*  100

  11* D
  12*  500
```

Example 11

As mentioned, digits in numeric values for textual items are treated as characters for sorting purposes. Therefore, those values are not sorted numerically. For example, if STREET NUMBER (C1) and STREET NAME (C2) are both CHARACTER items, this command produces the following output:

```
LIST C1, C2, ORDERED BY C2, C1 WHERE C1 EXISTS:

* STREET NUMBER   STREET NAME
***
* 1617            MORROW
* 415             MORROW
* 901             MORROW
* 104             WOODHILL
* 2105            WOODHILL
* 602             WOODHILL
```

However, if STREET NUMBER (C2) is an INTEGER item, the same command produces this result:

```
* STREET NUMBER   STREET NAME
***
*           415   MORROW
*           901   MORROW
*          1617   MORROW
*           104   WOODHILL
*           602   WOODHILL
*          2105   WOODHILL
```

Chapter 17

Displaying Values Together: By-Clause

Format 17-2
Considerations 17-2
Examples 17-3

A retrieval-clause can also include a by-clause. A by-clause allows you to specify that only values for specified items within a data tree are to be displayed together in logical order. A by-clause allows you to

- specify exactly which values from a data tree should be retrieved and which should be omitted
- retrieve data from records that are disjoint from the qualified records
- retrieve data from disjoint records when the command includes a where-clause
- retrieve data in logical order when the command includes a where-clause
- specify that computations be performed on specified subsets of data, rather than on the entire database.

A by-clause causes logical order retrieval of specified data items in data trees topped by occurrences of the record specified in the by-phrase.

A by-phrase consists of the keyword BY followed by either a schema record or by the keywords DATA BASE. BY DATA BASE qualifies the entire database for processing; therefore, it causes logical order retrieval of all objects specified in the by-clause. Objects specified after BY DATA BASE by-phrases are displayed before values for items specified elsewhere in the retrieval-clause.

For example, the following command produces output with all values for C1 displayed first, followed by C2 and C3 values for each male employee:

```
PRINT C2, C3, BY DATA BASE, C1 WHERE C7 EQ MALE:
```

Format

```
by-phrase, units

unit  by-phrase
```

```
by-phrase   BY |record
               |DATA BASE

     unit   |item
            |record
            |* function *
            |* function (parameters)
            |* string *
            |* string (parameters)
            |(adhoc-function)
            |system-function
            |cf-item
```

item is a schema item name or component number.

record is a schema record name or component number.

function is a stored function name or component number.

string is a stored string name or component number.

parameters are the values that replace the numbered parameters in the stored function or string.

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions" for details.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. You can use numeric items only with AVG, SIGMA, and SUM. And you can use only records with COUNT. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions" for details.

cf-item is a Collect File item name.

Considerations

- If the type of record specified in a by-phrase is an ancestor of the qualified data records, the command may yield unwanted results. This is because data trees with the focal record at the top may contain records with values that do not satisfy the conditions in the where-clause.

- You can specify a by-clause in the retrieval-clause of a COLLECT command. The command is processed the same way as other retrieval commands.
- See **Specifying the Focal Record for a System Function** on page 34-6 for a discussion of the use of BY with system functions.

Examples

Example 1

Notice that C102 and C132 are items in the tree topped by record C100. *BY C100* is the by-phrase, and *BY C100, C102, C132* is the by-clause in the following command:

```
LIST C1, C2, BY C100, C102, C132:
```

In the next command, the target record (C100) is related to C0, the type of record qualified by the where-clause. Also, C1 and C2 are in record C0, which is in the path of C100. *BY C100* is the by-phrase, and *BY C100, C102, C132* is the by-clause.

```
LIST C1, C2, BY C100, C102, C132 WHERE C7 EQ FEMALE:
```

In the next command, record (C100) is related to C0, the type of record qualified by the where-clause. Also, C1 and C2 in the retrieval-clause and C2 in the ordering-clause are in record C0, which is in the path of C100. *BY C100* is the by-phrase, and *BY C100, C102, C132* is the by-clause.

```
LIST C1, C2, BY C100, C102, C132, ORDERED BY C2 WHERE C7 EQ FEMALE:
```

In the next command, all items and records are in the same path. *BY C100* and *BY C110* are by-phrases, and *BY C100, C102, C132* and *BY C110, C121* are by-clauses.

```
LIST C1, C2, BY C100, C102, C132, BY C110, C121 WHERE C7 EQ FEMALE:
```

Example 2

This command uses a by-clause to display only specified values from a data tree. The command displays only values for C102, C111, and C122 in data trees topped by selected C100 records.

```
PRINT C1, C2, C3, BY C100, C102, C111, C122 WHERE C7 EQ FEMALE:
```

In contrast, the following command displays all values in data trees topped by selected C100 records:

```
PRINT C1, C2, C3, C100 WHERE C7 EQ FEMALE:
```

Example 3

This command uses a by-clause to display values for items in disjoint records C100 and C200.

```
PRINT BY C0, C102, C201 WHERE C7 EQ FEMALE:
```

If the above command is issued without a by-clause, as shown below, the software cannot process the command because a where-clause is present and the retrieval items are in disjoint records. Instead of processing the command, the software issues a message.

```
PRINT C102, C201 WHERE C7 EQ FEMALE:

-326- ACTION-CLAUSE COMPONENTS MUST BE IN ONE PATH -
```

That restriction does not apply to commands without a where-clause, so the following command can be processed:

```
PRINT C102, C201:
```

Example 4

This command uses a by-clause to display values in logical order. Without a by-clause, the output would appear in physical order. The values for C1 and C102 would appear according to the Hierarchical Table address of the selected C100 records.

```
PRINT BY ENTRY, C1, C102 WHERE C7 EQ FEMALE:
```

Example 5

This command uses by-clauses for calculations. In this case, BY DATA BASE causes the software to count the number of entries in the entire database. (In the output, 16 is the number of female employees, and 51 is the total number of entries in the database.)

```
PRINT COUNT C0, COUNT C0 BY DATA BASE WHERE C7 EQ FEMALE:

CNT 0* 16
CNT 0* 51
```

Example 6

This example uses a system function that has a system function as its object. In this case, the software computes the sum of the values in GROSS PAY for each entry and then computes the average of all the SUMs, yielding a single value.

```
PR AVG (SUM C124 BY C0):

AVG 10001* $14,167.89
```

Chapter 18

Retrieval Examples by Task

Index to Examples

Commands and Syntax Units	Example Numbers
Ad hoc functions	14
By-clause	11 12 13
DITTO	4 6
Format options	6 8 12 18
LIMIT	19
LIST (without layout specifications)	7 8 9 10 11 12 15 16 17 20
LIST (with layout specifications)	13 14 18
Ordering-clause	7 8 9 10 11 13 14 16 18 20 21
PRINT	1 2 3 4 5 6
SAME	6
SAME IS STATIC/DYNAMIC	6
System functions	3 4 5 6
System strings (*NOW*, *FTODAY*)	14
TALLY	5 19
Where-Clause	
AT operator	3 4 7 18 20 21
Binary operators:	
EQ	1 2 3 4 6 7 11 12 13 18
NE	9
LE, GE, LT, GT	4 8 10 11 17 18
item-to-item comparison	18
CONTAINS	15, 16
HAS operator	18, 21
Logical operators: AND, OR	2 4 6 11 12 13 17 18 20 21
NOT operator	13
Parentheses	11 13 14 18 21
Ternary operators: EQ, NE, SPANS	5
Unary operators:	
EXISTS	13 14 20 21
FAILS	17 20 21

Example 1

You need the name of the employee whose employee number is 1112.

```
PRINT LAST NAME WHERE C1 EQ 1112:

  2* JONES
```

Since employee numbers (C1) are unique, a single condition in the where-clause provides the requested information. Although additional information about this employee could be obtained by including other items in the retrieval-clause, the only information requested here is the LAST NAME (C2).

Example 2

You need the employee number of a person named Smith. You know that the employee's manager has the initials of ILP.

Although this request appears to be almost identical to Example 1, a command with a single item in the retrieval-clause and a single condition in the where-clause would produce simply the three employee numbers. Therefore, the information about this employee's manager must be used.

The following two commands show different approaches to obtaining the requested information:

```
PRINT EMPLOYEE NUMBER, C103 WHERE C2 EQ SMITH:

1*  1313
  103* AFG
1*  1010
  103* ILP
1*  1272
  103* RMJ

PRINT C1 WH LAST NAME EQ SMITH AND C103 EQ ILP:

1*  1010
```

The first command produces the EMPLOYEE NUMBER (C1) and the MANAGER (C103) for each of the employees with the LAST NAME (C2) of Smith. From this you can select the EMPLOYEE NUMBER associated with the appropriate value for MANAGER, in this case, 1010.

The second command includes two conditions joined by the logical operator AND: the LAST NAME must be Smith, and the MANAGER must be ILP. AND means that both conditions must be met in order for a data record to qualify. As shown, this command produces one EMPLOYEE NUMBER, 1010, since there is only one person in the EMPLOYEE database who has the LAST NAME of Smith and also has ILP as the value for MANAGER. In another database, you might need to know (or retrieve) FORENAME or some other value in order to isolate the needed employee number.

Example 3

You need the number of employees who currently have JBM as their manager. Since the information requested is simply a numerical total, the system function COUNT provides an efficient way of obtaining the desired data.

```
PRINT COUNT C0 WH C103 EQ JBM AT 1:

CNT 0*  4
```

In the command, the record ENTRY (C0) is used with COUNT, although any item (for example, EMPLOYEE NUMBER) or record that occurs only once for any employee could have been used. The C0 record is used with COUNT because processing is cheaper. When COUNT is used with an item, the software must reference the Hierarchical Table and the Data Table, checking for nulls and counting only non-nulls. However, when COUNT is used with a schema record, it does not reference the Data Table and is, therefore, cheaper to use.

The where-clause contains the condition that the value for MANAGER (C103) be JBM. Since an employee could have held more than one position with more than one manager and since the latest information in the EMPLOYEE database is always inserted in the left (or the AT 1) position in the data tree, the AT operator qualifies only those who currently have JBM as their manager.

Example 4

You need the number of females and males currently receiving a salary of $1,500.00 or more. Since this example, like Example 3, requests no information about individuals but merely the number of females and the number of males, the system function COUNT is also appropriate.

```
PR COUNT ENTRY WH C7 EQ FEMALE AND C124 GE $1500.00 AT 1:

CNT 0*  9

DITTO WH GENDER EQ MALE AND C124 GE $1500.00 AT 1:

CNT 0*  27
```

In this example, record C0 is used with COUNT in the retrieval-clause since processing is cheaper, as noted in Example 3. The information requires two commands because there is no way of qualifying the females who make $1500 or more and the males who make $1500 or more with a single where-clause and print the results separately. DITTO is used in the second command to repeat the retrieval-clause.

The where-clause contains the binary operator GE (greater than or equal to) since the values required for GROSS PAY (C124) are $1500.00 or more. The logical operator AND is used so that, in the first command, only those females making $1500 or more are qualified and, in the second command, only those males making $1500 or more are qualified. The operator AT with 1 is used because the information requested concerns current salaries, which, in the EMPLOYEE database, are inserted in the first position among siblings.

Example 5

You need the total federal tax deductions for April 1991.

```
PR SUM C125 WH C121 SPANS 04/01/1991*04/30/1991:

SUM 125* $15,739.24
```

This command uses the system function SUM in the retrieval-clause and the ternary operator SPANS in the where-clause. The SPANS operator qualifies all data records having any value between the two values specified as well as the given values, that is, any April 1991 value for C121. (Example 6 shows how the EQ operator can be used instead of the SPANS operator.) Then, using only the qualified data records, the software sums all the values for the item FEDERAL TAX DEDUCTION (C125). This command provides the requested information in a single step, even if you do not know what values are in C121, rather than the two or more steps required in the method described below.

Since C121 is a key item, a second method of obtaining the information would be to issue the following TALLY command:

```
TALLY/EACH/C121:

*****************************************
 ITEM-        PAYROLL MONTH
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
       46     01/31/1991
       47     02/28/1991
       47     03/30/1991
       48     04/30/1991
       50     05/31/1991
-----------------------------------------
        5 DISTINCT VALUES
-----------------------------------------
      238 TOTAL OCCURRENCES
-----------------------------------------
```

The TALLY command provides the distinct values for the item given, thus allowing you to select the exact dates stored for April 1991. In the case of the EMPLOYEE database, the only April 1991 value for C121 is 04/30/1991, so the following command would also provide the requested information:

```
PR SUM C125 WH C121 EQ 04/30/1991:

SUM 125* $15,739.24
```

Example 6

You need a report with the total gross pay, the average gross pay, the total net pay, and the average net pay for the entire company and for each department in 1991. This is a complex request. You need to obtain four figures for the entire company and for each of the four departments. By using DITTO and SAME, you can obtain the information for the entire company first, then add the condition for the department in subsequent where-clauses. DITTO reduces the time it takes to enter the commands and the possibility of errors. SAME has the same effects, plus it saves the reprocessing time for one or more where-clause conditions.

```
PR /NAME/ SUM C124, AVG C124, SUM C126, AVG C126 WH C121 EQ
   01/01/1991*12/31/1991:

SUM GROSS PAY* $708,394.42
AVG GROSS PAY* $2,976.45
SUM NET PAY* $590,030.39
AVG NET PAY* $2,479.12

SAME IS STATIC:
DITTO WH SAME AND C102 EQ /ADMINISTRATION & FINANCE/:

SUM GROSS PAY* $56,254.20
AVG GROSS PAY* $1,520.38
SUM NET PAY* $47,601.84
AVG NET PAY* $1,286.54

DITTO WH SAME AND C102 EQ EXECUTIVE:

SUM GROSS PAY* $166,470.36
AVG GROSS PAY* $3,329.41
SUM NET PAY* $137,506.81
AVG NET PAY* $2,750.14

DITTO WH SAME AND C102 EQ INFORMATION SYSTEMS:

SUM GROSS PAY* $197,137.50
AVG GROSS PAY* $2,464.22
SUM NET PAY* $167,486.00
AVG NET PAY* $2,093.57

DITTO WH SAME AND C102 EQ MARKETING:

SUM GROSS PAY* $288,532.36
AVG GROSS PAY* $4,063.84
SUM NET PAY* $237,435.74
AVG NET PAY* $3,344.17

SAME IS DYNAMIC:
PRINT/NUMBER/:
```

Example 6 (continued)

The retrieval-clause of the first command includes two system functions SUM and AVG for each of the items GROSS PAY (C124) and NET PAY (C126). Although the items are ordered to place the average of an item immediately below the total of that item, note that the system function must be repeated with each item requiring a system function. In other words, the following command would print a single value (the sum) for C124 and all the C126 values for selected records. There would be no sum produced for C126.

```
PR SUM C124, C126 WH ... :
```

The where-clause in the main command uses EQ as a ternary operator. EQ is equivalent to the SPANS operator when two values in the condition are separated by an asterisk (*). An equivalent way of writing the SPANS condition in a where-clause would be as follows:

```
WH C121 GE 01/01/1991 AND C121 LE 12/31/1991
```

However, this where-clause is more costly to process than the where-clause with EQ (or SPANS). Since the EMPLOYEE database is updated only through May 31, 1991, the following where-clause would obtain the same information:

```
WH C121 GE 01/01/1991
```

To obtain the data for each department, the qualification for DEPARTMENT (C102) must be added to the where-clause. The default for the SAME operator is DYNAMIC, which causes the software to use the records qualified by the where-clause in the immediately preceding command. Since the where-clause obtaining data about the year 1991 is what you want repeated, along with different values for an additional condition, a command changing SAME to STATIC is needed. This command tells the software to use the more recent of the where-clauses immediately preceding the SAME IS STATIC command or the last where-clause that did not contain a SAME operator.

DITTO is used in each retrieval-clause since the same information is desired for each of the departments.

Without DITTO and SAME, the commands for each department would be as follows, changing only the value for C102 for each of the four departments:

```
PR SUM C124, AVG C124, SUM C126, AVG C126 WH C121 EQ 01/01/1991*
   12/31/1991 AND C102 EQ /ADMINISTRATION & FINANCE/:
```

If you were not sure how the values for DEPARTMENT (C102) were stored in the database, issuing a TALLY/EACH/C102 command would provide a listing of this information.

If you want SAME to be DYNAMIC in subsequent commands, you can issue a SAME IS DYNAMIC command once all data have been obtained.

Example 7

You need a list of employees in the Information Systems Department, showing full names and position titles. You could obtain the information with a PRINT command, but using LIST provides the convenient and easy-to-use columnar format. The ordering-clause OB C2 causes the values to be printed in alphabetical order by LAST NAME (C2).

```
LI C2, C3, C101, OB C2 WHERE C102 EQ INFORMATION SYSTEMS AT 1:

* LAST NAME    FORENAME            POSITION TITLE
***
*                                  PROGRAMMER
* CAHILL       JACOB               MANAGER SYSTEM
                                   S DEVELOPMENT
                                   & MAINTENANCE
* FERNANDEZ    SOPHIA              STANDARDS & PR
                                   OCEDURES ANALY
                                   ST
* FREEMAN      LEOPOLD             SR SYSTEMS PRO
                                   GRAMMER
* GARCIA       FRANCISCO           JR PROGRAMMER/
                                   ANALYST
* GIBSON       MOLLY I.            TECHNICAL WRIT
                                   ER
* HERNANDEZ    JESSE L.            MANAGER DATA B
                                   ASE ADMINISTRA
                                   TION
* JOHNSON      BRADFORD            JR SYSTEMS PRO
                                   GRAMMER
* JONES        MICHAEL Y.          SR SYSTEMS ANA
                                   LYST
   .
   .
   .
```

Note that the first value under POSITION TITLE, in this case PROGRAMMER, has no corresponding values for LAST NAME or FORENAME (C3). This represents the entry for a position that is not filled.

The three items in the retrieval-clause appear in the output as three columns headed by the item names since there are no column heading specifications. The where-clause contains the condition that the value for DEPARTMENT (C102) be INFORMATION SYSTEMS. The AT 1 ensures that only those employees currently in the Information Systems Department be included since the EMPLOYEE database has the latest information in the first position among siblings.

Example 8

You need a list of employees who have been with the company for two or more years, including full names, employee numbers, and departments.

A LIST command gives easier readability. Since the EMPLOYEE database is updated only through May 31, 1991, the where-clause is written as though the information were requested at the end of May 1991. The C0 record of each employee who has a value for HIRE DATE (C4) that is less than the value specified in the command, 05/31/1989, is qualified by the where-clause. Using an ordering-clause with multiple sort keys causes the software to alphabetize by department, then alphabetize by last name within each department, and, finally, by first name, if applicable.

```
LI /REPEAT SUPPRESS/ C1, C2, C3, C102, OB C102, C2, C3
   WH C4 LE 05/31/1989:

* EMPLOYEE NUMBER   LAST NAME    FORENAME          DEPARTMENT
***
*            1071   CANADY       FRANK A.          ADMINISTRATION
                                                     & FINANCE
*            1112   JONES        RITA M.           ADMINISTRATION
                                                     & FINANCE
*            1086   SCHMIDT      PENNY             ADMINISTRATION
                                                     & FINANCE
*            1002   BOWMAN       HUGH E.           EXECUTIVE
*            1105   FAULKNER     CARRIE ANN        EXECUTIVE
*            1006   GARRETT      OLAN M.           EXECUTIVE
*            1005   KNAPP        PATRICE R.        EXECITOVE
*            1004   KNIGHT       ALTHEA            EXECUTIVE
*            1074   MILLSAP      JOEL B.           EXECUTIVE
*            1024   MUELLER      PATSY             EXECUTIVE
*            1101   NATHANIEL    DARRYL            EXECUTIVE
*            1003   SALAZAR      YOLANDA           EXECUTIVE
*                                                  EXECUTIVE
*            1001   WATERHOUSE   CLIFTON P.        EXECUTIVE
*            1067   CAHILL       JACOB             INFORMATIO SY
                                                     STEMS
*            1049   FERNANDEZ    SOPHIA            INFORMATIO SY
                                                     STEMS
*            1008   HERNANDEZ    JESSE L.          INFORMATIO SY
                                                     STEMS
*            1009   JONES        MICHAEL Y.        INFORMATIO SY
                                                     STEMS
*            1092   POLANSKI     IVAN L.           INFORMATIO SY
                                                     STEMS
   .
   .
   .
```

Since there are no column heading specifications, the item names are printed as column headings, and the column widths are the number of characters in the item name or the default width, whichever is larger.

The REPEAT SUPPRESS format option is included in the command so that duplicate information (that is, EMPLOYEE NUMBER and NAME) is not displayed for those employees who have been with the company since May 1989.

Example 9

You need a list of employees who are not Caucasian, including full names and ethnic origin.

As in Examples 7 and 8, LIST will be used instead of PRINT for easier viewing. To obtain ethnic origins of employees who are not Caucasian, it might seem that the appropriate where-clause is WHERE NOT C8 EQ CAUCASIAN. However, the logical operator NOT would cause the software to qualify data records with null values for ETHNIC ORIGIN (C8).

In the EMPLOYEE database, there is one logical entry for an unfilled position that would have been included in the output if the above-mentioned where-clause were used. The binary operator NE (not equal) is used instead, because it causes the software to qualify only those records that have a value other than CAUCASIAN. The ordering-clause with multiple sort keys causes the software to alphabetize by ethnic origin, by last name within ethnic groups, and finally by first name.

```
LI C2, C3, C8, OB C8, C2, C3 WH C8 NE CAUCASIAN;

* LAST NAME    FORENAME              ETHNIC ORIGIN
***
* LITTLEJOHN   FANNIE                AMERICAN INDI
                                     AN
* REDFOX       RICHARD B.            AMERICAN INDI
                                     AN
* AMEER        DAVID                 ASIAN
* CHAN         TAI                   ASIAN
* BROOKS       RUBEN R.              BLACK
* BROWN        VIRGINIA P.           BLACK
* GIBSON       GEORGE J.             BLACK
* GIBSON       MOLLY I.              BLACK
* JOHNSON      BRADFORD              BLACK
* SEATON       GARY                  BLACK
* SMITH        JERRY LEE             BLACK
* FERNANDEZ    SOPHIA                HISPANIC
* GARCIA       FRANCISCO             HISPANIC
* HERNANDEZ    JESSE L.              HISPANIC
* JONES        RITA M.               HISPANIC
* JUAREZ       ARMANDO               HISPANIC
* QUINTERO     PEDRO                 HISPANIC
* RODRIGUEZ    ROMUALDO R.           HISPANIC
* SALAZAR      YOLANDA               HISPANIC
```

If you knew all values in the database for item ETHNIC ORIGIN, then the binary operator OR could have been used, although this would require more processing time and a longer where-clause, as shown below:

```
WH C8 EQ AMERICAN INDIAN OR C8 EQ ASIAN OR C8 EQ BLACK
   OR C8 EQ HISPANIC
```

You could also use the IN operator, as shown below:

```
WHERE C8 IN (AMERICAN INDIAN, ASIAN, BLACK, HISPANIC)
```

Since ETHNIC ORIGIN is a key item, a TALLY/EACH/C8 command identifies the distinct values found in the database for ETHNIC ORIGIN, thus providing the values to be used in the where-clause above.

Example 10

You need a list of employees who have changed positions in the last six months (from 12/01/1990), including employee numbers, last names, position titles, departments, and start dates.

Since the information requested concerns the starting date of the present position, the AT operator is needed because the EMPLOYEE database was designed to contain the latest information in the AT 1 position.

```
LI C1, C2, C101, C102, C105, OB C102 WH C105 GT 12/01/1990
   AT 1 AND C105* GT C4*:

* EMPLOYEE NUMBER   LAST NAME    POSITION TITLE    DEPARTMENT        START DATE
***
*            1006   GARRETT      VICE-PRESIDENT    EXECUTIVE         01/01/1991
                                  MARKETING
*            1009   JONES        SR SYSTEMS ANA    INFORMATION SY    01/01/1991
                                  LYST              STEMS
*            1077   GIBSON       INSTRUCTOR        MARKETING         01/01/1991
```

The second condition, C105* GT C4*, excludes those employees who were hired in the last six months, since they have not actually changed positions. That is, the condition specifies that the value for START DATE (C105) in the current position must be greater than the value for HIRE DATE (C4).

An asterisk (the default system separator) must follow each item number when two items are compared with a binary operator. As you would expect, only numeric items can be compared with numeric items and only alphabetic items can be compared with alphabetic items.

This second condition, C105* GT C4*, is treated as a non-key condition. The logical operator AND means that both conditions included in the where-clause must be met.

Since no column headings are specified, the width of each column is determined by the default width or the number of characters in the item name, whichever is larger. The column headings default to the item names.

Example 11

You need a list of employees who have completed a graduate degree in the last five years, including last names, position titles, departments, gender, and pay rates.

PRINT could provide this information, but LIST will present the data in a more readable format.

This example requires several conditions in the where-clause, one concerning the date the graduate degree was completed and one each for the different graduate level degrees. Since the request asks for any graduate degree, the IN operator implies several conditions connected with the OR operator. (The TALLY command for C412 provides a listing of the distinct values stored in the DEGREE/CERTIFICATE item.)

```
LI BY ENTRY, C2, C101, C102, C7, PAY, RATE OB C2
   WH C413 GT 06/01/1986 AND (C412 IN (MA, MBA, MS, PHD):
```

* LAST NAME ***	POSITION TITLE	DEPARTMENT	GENDER	PAY RATE
* CHAN	SR SALES REPRESENTATIVE	MARKETING	MALE	$3,300.00
*				$3,150.00
*				$2,850.00
*				$2,700.00
* GARCIA	JR PROGRAMMER/ ANALYST	INFORMATION SYSTEMS	MALE	$1,800.00
* POLANSKI	MANAGER SYSTEMS OPERATIONS & RESOURCES	INFORMATION SYSTEMS	MALE	$3,900.00
*				$3,600.00
*				$3,375.00
* SMITH	JR SALES REPRESENTATIVE	MARKETING	MALE	$2,700.00

When two conditions are connected with the OR operator, either one condition or both must be satisfied.

Parentheses are used so that the software collects all records with the specified degrees before it checks whether those records also contain the desired value for date of completion. Without parentheses, the software would have first qualified those records with both C413 GT 06/01/1986 and C412 EQ MA since the AND operator is processed first. Then, the software would have also chosen those records with the other values for C412 regardless of the value for C413 found within those records. An employee with an MBA obtained prior to 06/01/1986 could have been listed.

BY ENTRY normalizes upwards in the data tree because the conditions in the where-clause are in a different path from the items included in the retrieval-clause. (C412 and C413 are in a different path than C101 and C102).

The ordering-clause OB C2 is included to alphabetize the employees by LAST NAME (C2).

Example 12

You need a report showing the employees with skills in a programming language, COBOL, FORTRAN, PASCAL, or PL/1. The report needs to provide last names, position titles, extension numbers, and departments. The LIST command will present the data in a more readable format, which is especially important when a large amount of data is to be displayed. The IN operator creates four conditions connected with the OR operator.

```
LI /REPEAT SUPPRESS/ C2, C10, C201, BY ENTRY, C101, C102
   WH C201 IN (COBOL, FORTRAN, PASCAL, PL/1):

* LAST NAME    OFFICE-EXTENSION   SKILL TYPE   POSITION TITLE     DEPARTMENT
***
* REID         410 XT369          PL/1         PROGRAMMER         INFORMATIO SY
                                                                  STEMS
*                                                ASSISTANT PROG     INFORMATIO SY
                                                 RAMMER             STEMS
*                                 COBOL
*                                 FORTRAN
* WATERHOUSE   101 XT109          COBOL        PRESIDENT          EXECUTIVE
* SCHOLL       318 XT419          FORTRAN      SR SALES REPRE     MARKETING
                                                 SENTATIVE
*                                                JR SALES REPRE     MARKETING
                                                 SENTATIVE
* BROOKS       581 XT347          PASCAL       JR SALES REPRE     MARKETING
                                                 SENTATIVE
* AMEER        545 XT495          PASCAL       SR SALES REPRE     MARKETING
                                                 SENTATIVE
*                                                JR SALES REPRE     MARKETING
                                                 SENTATIVE
* GOODSON      323 XT512          PASCAL       SR SALES REPRE     MARKETING
                                                 SENTATIVE
* WILLIAMSON   218 XT802          COBOL        INSTRUCTOR         MARKETING
*                                 COBOL
   .
   .
   .
```

The where-clause contains four conditions joined by the logical operator OR. OR means that the condition before it must be met, the condition after it must be met, or both conditions must be met.

The REPEAT SUPPRESS format option is included so that duplicate information (such as, LAST NAME (C2), POSITION TITLE (C101), OFFICE-EXTENSION (C10), and DEPARTMENT (C102)) is not printed for those employees having more than one programming language skill. Without the REPEAT SUPPRESS option, any employee having two or more programming skills would have all items listed in the retrieval-clause repeated for each programming skill. In other words, the software selects those C200 records having one of the values listed in the where-clause, then goes back up the path to the C0 record to print from each selected record unless the REPEAT SUPPRESS option is in effect.

The BY ENTRY is required since there are two different paths represented in the retrieval-clause. BY ENTRY, in this case, normalizes the C100 records up to level 0 so that they will be in the same path as the item in the C200s (C201). Notice that items in all occurrences of C100 are listed, not just the most recent.

Example 13

You need a report showing a list of employees who do not have a college degree, showing their full names, departments, and position titles.

```
LI /TITLE LAST  NAME, FIRST NAMES, L(13)DEPARTMENT,POSITION TITLE,
   L(19)DEGREE-CERTIFICATE/ BY ENTRY, C2, C3, C102, C101, C412,
   OB C412, C2 WH C412 EXISTS AND NOT (C412 EQ BA OR C412 EQ BS
-339- WARNING- ORDERING-CLAUSE COMPONENTS BELOW RETRIEVAL COMPONENTS -
   OR C412 EQ MA OR C412 EQ MBA OR C412 EQ MS OR C412 EQ PHD):

* LAST  NAME   FIRST NAMES    DEPARTMENT      POSITION TITLE  DEGREE-CERTIFICATE
***
* NATHANIEL    DARRYL         EXECUTIVE       SECRETARY
*                                                            AA
* SALAZAR      YOLANDA        EXECUTIVE       ADMINISTRATIVE
                                                ASSISTANT
*                             EXECUTIVE       SECRETARY
*                                                            AA
* SMITH        GARLAND P.     ADMINISTRATIO   BOOKKEEPER
                              N & FINANCE
*                                                            AA
* WAGGONNER    MERRILEE D.    MARKETING       SECRETARY
*                                                            AA
* WILLIAMSON   JANICE L.      MARKETING       INSTRUCTOR
*                                                            AA
*                                                            BA
* COLLINS      LILLIAN        ADMINISTRATIO   MAIL CLERK
                              N & FINANCE
*                                                            HIGH SCHOOL DIPLOMA
* KAATZ        FREDDIE        ADMINISTRATIO   SUPPLY CLERK
                              N & FINANCE
*                                                            HIGH SCHOOL DIPLOMA
* LITTLEJOHN   FANNIE         MARKETING       SECRETARY
*                                                            HIGH SCHOOL DIPLOMA
* MUELLER      PATSY          EXECUTIVE       SECRETARY
*                                                            HIGH SCHOOL DIPLOMA
* SCHMIDT      PENNY          ADMINISTRATIO   SECRETARY
                              N & FINANCE
*                                                            HIGH SCHOOL DIPLOMA
* SLYE         LEONARD R.     ADMINISTRATIO   GENERAL MAINTEN
                              N & FINANCE     ANCE
*                                                            HIGH SCHOOL DIPLOMA
```

This example shows several conditions, all of which are based on the item DEGREE/CERTIFICATE (C412).

- The binary operator OR means that the condition before the OR must be met, the condition after the OR must be met, or both conditions must be met.

- Parentheses are included around the conditions connected by OR since the software collects all records meeting the conditions stated within parentheses before executing the NOT operator. In this case, when the software executes the NOT operator, it qualifies all records that do not meet the conditions stated within parentheses, including any null values.

(continued next page)

Example 13 (continued)

- The only null value for C412 is in the logical entry for the unfilled position. Since including this entry would produce -NULL- in each of the columns, the first condition, C412 EXISTS, is included.

- The unary operator EXISTS qualifies all records having a value for the given item but does not qualify nulls.

- The binary operator AND means that both conditions it joins must be met. Thus, the only records that qualify are those where a value exists for C412 and that value is not one of those listed within parentheses.

This example also includes column heading and column width specifications. These occur within the slashes that begin after the keyword LIST and end after the last column heading specification.

- The column headings correspond one-to-one with the items in the retrieval-clause.

- The first, second, and fourth headings do not have column width specifications; therefore, the width is the default width or the column heading given, whichever is larger.

- Since all items in the retrieval-clause are CHARACTER, the headings are left-justified.

- Since there are no B-specifications for blanks between the columns, three spaces separate columns of data, by default.

- The L(13) before DEPARTMENT tells the software to left-justify the column heading and to make the column width 13 spaces wide.

- The L(19) before DEGREE-CERTIFICATE tells the software to left-justify the column heading and to make the column 19 spaces wide. (Although a slash may be more appropriate between the words DEGREE and CERTIFICATE, the slash cannot be used because it is reserved to set off the title, paging, and column heading specifications in a LIST command.)

BY ENTRY normalizes the C100 records up to level 0 and is required since the items in the retrieval-clause fall into two paths. The normalization to level 0 causes the fifth entry to be included in the output even though that person has a BA.

The ordering-clause sorts the output alphabetically by DEGREE/CERTIFICATE and then by LAST NAME within each of these. Note the ordering-clause warning message immediately below the third line of the command. Although not a fatal error, this message warns that ordering at a level lower than what is to be printed causes items to be printed for each selected record, as ordered. For example, if the first entry listed had a BA degree in addition to the AA, this entry would have appeared again between the last AA entry and the first HIGH SCHOOL DIPLOMA entry.

Example 14

You need a list of employees with full names, current ages, and ages when hired. This example requests a listing of all employees, so the only qualification is that there be a last name; that is, that the value for LAST NAME (C2) is not null. The unary operator EXISTS causes only those records with a value for LAST NAME to be qualified.

```
LI /TITLE LAST  NAME, FIRST NAMES, R(8)EMPLOYEE+AGE  ,
   R(8)HIRE AGE/ C2, C3, ((*FTODAY*-C5)/365.25),
   ((C4-C5)/365.25), OB C2 WH C2 EXISTS:

* LAST  NAME  FIRST NAMES    EMPLOYEE   HIRE AGE
                              AGE
***
* AMEER       DAVID           27.847     24.643
* BOWMAN      HUGH E.         48.088     44.468
* BROOKS      RUBEN R.        27.469     27.179
* BROWN       VIRGINA P.      33.227     29.607
* CAHILL      JACOB           38.637     35.031
* CANADY      FRANK A.        37.736     34.366
* CHAN        TAI             33.114     29.832
* COLLINS     LILLIAN         28.290     27.318
* FAULKNER    CARRIE ANN      27.995     25.383
* FERNANDEZ   SOPHIA          34.924     31.680
* FREEMAN     LEOPOLD         44.512     42.727
* GARCIA      FRANCISCO       24.279     23.242
* GARRETT     OLAN M.         44.559     40.939
* GIBSON      MOLLY I.        29.752     28.882
* GIBSON      GEORGE J.       33.311     30.442
* GOODSON     ALAN F.         29.150     26.576
* HERNANDEZ   JESSE L.        46.387     42.768
   .
   .
   .
```

The command includes column heading specifications, primarily because the two ad hoc functions in the retrieval-clause would have no column headings unless one is specified for each. Since column headings cannot be specified for the third and fourth items without specifying headings for the first two, headings are given for the two columns of employee names. Other features of the column heading specifications are as follows:

- To make the column width for the first heading equal to ten, so that most last names would fit in one line without wraparound, two spaces are placed between LAST and NAME. A column width of ten could also be obtained by specifying L(10)LAST NAME, or R(10)LAST NAME.

- The second column is eleven spaces wide because the column heading specification contains ten letters and one space.

- The R(8) at the beginning of the third column heading specification causes the software to right-justify EMPLOYEE and AGE in a column width of eight.

- The + between the words EMPLOYEE and AGE causes the word following the + to be printed on the line below EMPLOYEE.

(continued next page)

Example 14 (continued)

- The three spaces between AGE and the comma center AGE under EMPLOYEE. The spaces are before the comma since this column is right-justified.

- The fourth column is also right-justified and is eight spaces wide because of the R(8) specification.

There are two ad hoc functions in the retrieval-clause. The first one includes the system string *FTODAY*, which provides the current date. Subtracting the BIRTHDAY (C5) value gives the number of days that have elapsed between the birthday and the day the command is issued. The slash means "divided by"; the 365.25 is a close approximation to the actual number of days per year. This division is carried to three decimal places by default.

The second ad hoc function (and fourth item in the retrieval-clause) subtracts the BIRTHDAY value from the HIRE DATE (C4) value for each employee to give the employee's age in days on the date hired. Dividing by 365.25 provides the number of years of age on the HIRE DATE.

Note that in both ad hoc functions, the numerator (that part that is to be divided by 365.25) must be enclosed with parentheses, and the entire ad hoc function must be enclosed in parentheses.

The ordering-clause prints the output in alphabetical order by LAST NAME.

Example 15

You need a list of employees with middle initial D, including full names and employee numbers. This command uses the CONTAINS operator with the WORD option. WORD causes the software to look in each C3 (FORENAME) value for the letter D where it is preceded and followed by at least one blank or special character, that is, any character less than hexadecimal 81. See Appendix B for the collating sequence of characters.

```
LI C1, C2, C3 WH C3 CONTAINS THE WORD D:

* EMPLOYEE NUMBER   LAST NAME    FORENAME
***
*            1017   WAGGONNER    MERRILEE D.
*            1083   SAVAGE       WILLIAM D.
*            1156   REED         KENNETH D.
```

Example 16

You need a list of employees with phone extensions in the 100's, including their full names, office numbers, and phone extensions.

```
LI C2, C3, C10, OB C2 WH C10 CONTAINS THE CHARACTERS 100*199
   WITH MATCH ON 3 CHARACTERS FROM POSITION 7 THROUGH LAST:

* LAST NAME     FORENAME              OFFICE-EXTENSION
***
* KNAPP         PATRICE R.            222 XT 123
* SALAZAR       YOLANDA               111 XT169
* SLYE          LEONARD R.            140 XT123
* SMITH         JERRY LEE             327 XT169
* WATERHOUSE    CLIFTON P.            101 XT109
```

This command uses the CONTAINS operator with the MATCH option. The software searches each C10 (OFFICE-EXTENSION) value starting in position 7 for 3 characters that match one of the numbers between 100 and 199 (inclusive). If all values for C10 had the same format (that is, no blanks between XT and the extension number), the condition could be specified as follows:

```
C10 CONTAINS XT100*XT199
```

The conditions used in the command can be abbreviated as shown here:

```
C10 CONT 100*199 MATCH 3 FROM 7
```

Example 17

You need a list of employees hired in 1991 who do not have a security clearance.

```
LI C2, C3 WH C13 FAILS AND C4 >= 01/01/91:

* LAST NAME     FORENAME
***
* SLYE          LEONARD R.
* BROOKS        RUBEN R.
* SMITH         JERRY LEE
```

The condition C4 GE 01/01/91 qualifies those employees hired in 1991, because the EMPLOYEE database is updated only through May 1991. If the database included data for 1992, then the condition could be specified using the SPANS operator:

```
C4 SPANS 01/01/91*12/31/91
```

The other condition in the where-clause uses the FAILS operator to identify those employees without a security clearance (C13). This condition is treated as a non-key condition because of the FAILS operator.

The where-clause can also be written by using other operators or switching the conditions. Some of the ways the where-clause can be constructed are as follows:

```
(1)  WH C13 FAILS AND C4 GE 01/01/91
(2)  WH NOT C13 EXISTS AND C4 GE 01/01/91
(3)  WH C4 GE 01/01/91 AND NOT C13 EXISTS
(4)  WH C4 GE 01/01/91 AND C13 FAILS
```

All these are equivalent; that is, they qualify the same data records. However, as discussed below, because they are not processed the same way, one where-clause can be more efficient than the others.

The software processes the first where-clause by making a list of the Hierarchical Table addresses of the C0 records with C4 GE 01/01/91, then examining each record in the Data Table for a C13 value. If no C13 value is found, the record qualifies. The software processes the second and third where-clauses by making two lists: the Hierarchical Table addresses of the C0 records with C4 GE 01/01/91 and the Hierarchical Table addresses of the C0 records with C13 values. The qualified records for either where-clause are determined by comparing the two lists: those Hierarchical Table addresses that appear only in the first list are the addresses of the qualified data records.

(continued next page)

Example 17 (continued)

The fourth where-clause is processed in the same way as the first; however, before producing the list of Hierarchical Table addresses, the software must first flip the conditions so that the key condition follows the non-key condition, that is, C13 FAILS AND C4 GE 01/01/91.

Which where-clause to use depends on the number of records satisfying each condition. If there are fewer records satisfying C4 GE 01/01/91 than satisfying C13 EXISTS, then the first where-clause should be used. Otherwise, either the second or third is more efficient. The last where-clause is the least efficient because it requires the software to switch the conditions before evaluating them.

Example 18

You need a list of employees who are currently working in the Marketing Department and who hold a college degree, including their full names, degrees, and major field of study. As before, the LIST command is used. The ordering-clause specifies C2 as a sort key. Because this can cause repetition of names for persons with multiple degrees, the REPEAT SUPPRESS format option is specified.

```
LI /REPEAT SUPPRESS, TITLE L(10)LAST NAME,L(12)FIRST NAMES,
   L(6)DEGREE, L(26)MAJOR FIELD/
   C2, C3, C412, C414, OB C2
   WH (C412 EQ BA*BS OR C412 GE MA)
   AND C0 HAS C102 EQ MARKETING AT 1:
```

```
* LAST NAME    FIRST NAMES    DEGREE   MAJOR FIELD
***
* AMEER        DAVID          BS       MATHEMATICS
* BROOKS       RUBEN R.       BS       ENGLISH
* BROWN        VIRGINA P.     BA       LIBERAL ARTS
* CHAN         TAI            BA       COMPUTER SCIENCE
*                             PHD      COMPUTER SCIENCE
* GIBSON       GEORGE J.      BA       EDUCATION
*                             MS       EDUCATION
*                             MS       EDUCATION
* GOODSON      ALAN F.        BA       PUBLIC BUSINESS ACCOUNTING
* JUAREZ       ARMANDO        BS       ELECTRICAL ENGINEERING
*                             MS       ELECTRICAL ENGINEERING
* RICHARDSON   TRAVIS Z.      BS       LIBERAL ARTS
* RODRIGUEZ    ROMUALDO R.    BS       ADVERTISING
* SCHOLL       MADISON A.     MS       MARKETING
*                             BS       PSYCHOLOGY
* SHROPSHIRE   LELAND G.      BA       ECONOMICS
* SMITH        JERRY LEE      BA       COMPUTER SCIENCE
*                             MA       COMPUTER SCIENCE
* VAN HOTTEN   GWENDOLYN      BA       MARKETING
* WILLIAMSON   JANICE L.      BA       EDUCATION
```

The where-clause consists of two expressions joined by the logical operator AND. The first expression uses two conditions to specify the criterion that an employee must have a college degree. The first condition uses EQ as a ternary operator; that is, EQ is used instead of SPANS. The second condition uses the binary operator GE to specify those values of C412 that are greater than or equal to MA, that is, MA and PHD. Notice that *greater*, in this case, applies to values greater than MA in alphabetical order. The two conditions are joined by the logical operator OR.

After the software processes the first expression, all C410 records with values for C412 of BA, BS, MA, or PHD qualify for further processing. The condition in the second expression identifies those C100 records with the value of MARKETING for C102. AT 1 operates on those records and finds the ones in first position relative to their siblings. The HAS operator is needed to specify C0 as the focal record for the expression, because C410 and C100 are disjoint records. Thus, the qualified records for the where-clause are those C410 records containing the specified values for C412 that are also descendants of C0 records whose leftmost (AT 1) descendants have the value of MARKETING for C102.

The command uses an L-specification for all column headings except DEGREE. The L-specifications left-justify the headings and specify wider columns for the values. For example, because L(22) MAJOR FIELD is used, a long value such as ELECTRICAL ENGINEERING fits on one line.

Example 19

You need to verify that each employee number is unique.

```
LIMIT 2,0:

TALLY C1:

*****************************************
 ITEM-        EMPLOYEE NUMBER
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
        0 DISTINCT VALUES
-----------------------------------------
        0 TOTAL OCCURRENCES
-----------------------------------------
```

Only employee numbers that have two or more occurrences would have been displayed, along with the number of times they occured. In this case, all values for C1 are unique.

Example 20

You need a list of employees who have held *exactly* two positions since they were hired, including employee numbers and the date when the employee changed positions. The LIST command is used because it offers a more readable format, and the ordering-clause specifies C1 as a sort key.

```
LI C1, C106, OB C1 WH C106 EXISTS AT 2 AT 0 OR C106 FAILS AT 2 AT 0:

* EMPLOYEE NUMBER     END DATE
***
*            1003   01/02/1989
*            1015   01/01/1989
*            1050   12/31/1989
*            1120   02/28/1989
*            1145   12/31/1989
```

The where-clause consists of two expressions joined by the logical operator OR. The first expression is satisfied by those C100 (POSITION WITHIN COMPANY) records that have a value for C106 (END DATE) in the second position (AT 2) and that are also the rightmost (AT 0) records under C0. The second expression is satisfied by those C100 records that do not have a value for C106 in the second position (AT 2) and that are also the rightmost (AT 0) C100 records under C0. Thus, the where-clause guarantees that the qualified C100 records are the second and also the last C100 records under C0, regardless of whether they have a value for C106.

The FAILS operator causes the second condition to be treated as non-key. Therefore, this where-clause is expensive to process.

Example 21

You need a list of employees who have held *at least* two positions since they were hired, including the dates when they changed positions. As before, the LIST command is used because its output is easy to read, and the ordering-clause specifies C1 and C105 as sort keys.

```
LIST C1, C105, C106, OB C1, C105 WH C0 HAS (C106 EXISTS AT 2 OR
   C106 FAILS AT 2):

* EMPLOYEE NUMBER   START DATE     END DATE
***
*            1003   01/01/1988   01/02/1989
*            1003   01/02/1989
*            1006   01/01/1988   12/31/1988
*            1006   01/01/1989   12/31/1990
*            1006   01/01/1991
*            1015   03/05/1988   01/01/1989
*            1015   01/01/1989
*            1050   06/01/1988   12/31/1989
*            1050   01/01/1990
*            1120   02/16/1988   02/28/1989
*            1120   03/01/1989
*            1145   05/02/1989   12/31/1989
*            1145   01/01/1990
```

The where-clause includes two expressions joined by the logical operator OR and preceded by the HAS operator. The first expression is satisfied by those C100 (POSITION WITHIN COMPANY) records that have a value for C106 (END DATE) in the second position (AT 2). The second expression is satisfied by those C100 records that do not have a value for C106 in the second position.

The FAILS operator causes the second condition in the where-clause to be treated as non-key. Therefore, this type of where-clause is expensive to process.

The qualified records for the where-clause are those C0 records that have at least two C100 descendants. The selected records are those C100 records that are descendants of the qualified C0 records.

Notice that for each employee there is a null for END DATE (C106) because there is no value for that item in the first (most current) C100 record.

Updating Data

Introduction

Updating Data Records

Updating Data Trees

Creating a Logical Unit of Work

Update Examples by Task

Chapter 19
Introduction

OPTIMIZATION CONSIDERATIONS FOR UPDATING 19-2

The following commands update data in a database:

- ADD
- ASSIGN
- CHANGE
- REMOVE
- ASSIGN TREE
- COPY TREE
- INSERT TREE
- MOVE TREE
- REMOVE TREE

The ADD, ASSIGN, CHANGE, and REMOVE commands update items in existing data records that are occurrences of a single schema record. These commands do not create or delete data records or subtrees; that is, they do not modify the structure of a database.

The ASSIGN TREE, COPY TREE, INSERT TREE, MOVE TREE, and REMOVE TREE commands modify the structure of a database by either removing or moving data trees, by inserting new data records, or by copying data trees from one position in the database to another.

Before you use these chapters for reference, be sure you have read Chapter 4, "Updating Data."

OPTIMIZATION CONSIDERATIONS FOR UPDATING

The following techniques are considerations for optimizing update commands.

1. When using the CHANGE or the ADD command to modify several item values in selected data records, use the form that specifies the schema record. For example, it is more efficient to issue the command

```
CH C0 EQ 14*2311 HANSFORD*15*AUSTIN TX*16*78753*END* WH C1 EQ 1043:
```

   than to issue the following three commands:

```
CH C14 EQ 2311 HANSFORD* WH C1 EQ 1043:
CH C15 EQ AUSTIN TX* WH SAME:
CH C16 EQ 78753* WH SAME:
```

   This technique does not apply to the ASSIGN command unless all values in the selected data records are to be modified.

2. The following techniques apply when you are deleting entries from a database.

   a) If a large number of records is to be removed, the following procedure can make the operation significantly less costly (less I/O and less CPU time). The technique makes use of the CONTROL language CREATE INDEX, REMOVE INDEX, and REORGANIZE ALL commands, which are explained in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*. These commands can be issued by the master password holder. The DBA password holder (with authority) can also issue the REORGANIZE command. The technique consists of the following sequence of commands:

```
      .
      .
      .
CONTROL:
REMOVE INDEX /LOSE/ items:
   (Items are key items in the records to be removed.)
QUEST:
REMOVE TREE record where-clause:
CONTROL:
CREATE INDEX items:
   (Items are key items in the records removed.)
REORGANIZE ALL:
      .
      .
      .
```

   This procedure deletes the indexes for the key items with only a few I/Os to the index and no I/Os to the Hierarchical Table or to the Data Table except to handle the overflow for key values, if any.

   Then, the REMOVE TREE command(s) accesses only the Hierarchical Table to accomplish the removals.

The CREATE INDEX command changes the Definition Table to reflect that the items are now changed back to key from non-key, then makes a pass through the Hierarchical Table to see whether any records exist for the items mentioned. If they were all removed, no other action is taken. If some still remain, the indexes are created for those items. This involves fewer I/Os than a reload operation. The object of the REORGANIZE ALL command is to reclaim the index space freed by the REMOVE INDEX command.

The amount of optimization achieved depends on the percentage of records removed versus the percentage still in the database. If all occurrences of records for one schema record (or several schema records) are removed, the reduction in I/O and CPU time is quite significant. However, it can be expensive if there are many key values that overflow.

b) If more than half the entries are to be removed, it is probably faster to use the RELOAD command with a where-clause identifying the entries to be retained. For example, to remove data for a past year, for instance, 1989, you could issue this command:

```
RELOAD WHERE C22 GE 01/01/90:
```

3. When possible, use the PREVIOUS operator to repeat the value stream given in an immediately preceding update command.

4. Where applicable, use the DITTO operator to repeat action-clauses in consecutive commands.

5. For moving a data tree from one position to another, use the MOVE TREE command. The MOVE TREE command changes logical pointers and does not require any changes in the physical structure of the Data Table or the index. Therefore, it is faster and more convenient than unloading a data tree and then doing a REMOVE TREE followed by an INSERT TREE. These commands involve actual deletion and insertion of the data records and their values.

Chapter 20

Updating Data Records

INTRODUCTION 20-2
- *Output From Update Commands 20-2*
- *Format Specifications and Other Syntax Considerations 20-3*
- *Trace Notation 20-3*
  - *Full trace 20-4*
  - *Partial trace 20-5*

REPLACING A NULL WITH A VALUE: ADD 20-6
- *Adding a Value for One Item 20-6*
- *Adding Values for Multiple Items in a Record 20-6*
- *Formats 20-7*
- *Considerations 20-9*

REPLACING EITHER A VALUE OR A NULL WITH A VALUE: ASSIGN 20-10
- *Assigning a Value for One Item in a Record 20-10*
- *Assigning Values for Several Items in a Record 20-10*
- *Formats 20-12*
- *Considerations 20-14*

REPLACING A VALUE WITH A VALUE: CHANGE 20-15
- *Changing a Value for One Item in a Record 20-15*
- *Changing Values for Several Items in a Record 20-15*
- *Formats 20-16*
- *Considerations 20-18*

REPLACING A VALUE WITH A NULL: REMOVE 20-19
- *Removing a Value for One Item in a Record 20-19*
- *Removing Values for Multiple Items in a Record 20-19*
- *Formats 20-20*
- *Considerations 20-20*

INTRODUCTION

The following commands update items in existing data records that are occurrences of a single schema record:

- The **ADD** command replaces null items with values.
- The **ASSIGN** command replaces null or valued items with new values.
- The **CHANGE** command replaces values with new values.
- The **REMOVE** command replaces values with nulls.

These commands do not create or delete data records or subtrees. In other words, they do not modify the structure of the database. To update subtrees and corresponding data records, see Chapter 21, "Updating Data Trees."

The database cycle number is incremented when each update command is successful. (This number is displayed in the output of the DESCRIBE command.)

Output From Update Commands

There is no output as a result of update commands. However, there are three types of messages that can occur as a result of an update command. They are

- an error message
- a message indicating the number of selected data records
- the -373- DATA BASE UNALTERED - message.

To verify the results of update commands, you can issue appropriate retrieval commands following the updates.

The database is marked internally if any update process completes abnormally, for example, because of fatal errors, time limit job abends, operator drops, or system drops. (Syntax errors do not cause damage to a database.) Subsequently, if you try to update a database damaged from a previous job, the software issues the following message:

```
-810- WARNING- DATA BASE DAMAGED.  NO UPDATES PERMITTED -
```

This message appears in the Message File of every job using a database damaged by a previous job abend. The message is also printed if you attempt to use an update command. However, you can still save, restore, describe, and retrieve from the database.

Format Specifications and Other Syntax Considerations

Format specifications for the ADD, CHANGE, and ASSIGN commands are identical except for the keywords ADD, CHANGE, and ASSIGN. The REMOVE command uses a different format. Since it only removes values, there is no need to specify new values.

In the ADD, CHANGE, and ASSIGN commands, component numbers and values to the left of the where-clause must each be followed by a system separator, for example, 1*1120* 2*REID*. Values and component labels in the where-clause are not followed by a system separator.

When you specify a new value for an item in an update command, the value must be coded to conform to the type and picture of the item. The only difference is in coding TEXT or UNDEFINED values after the EQ operator. In those cases, the blank space after EQ acts as a delimiter for the operator and does not become part of the value. For example, assuming that C132 is a TEXT or UNDEFINED item, the following command adds the value ABC:

```
ADD C132 EQ␣ABC* WHERE C1 EQ 2345:
```

However, the following command adds the value ␣ABC:

```
ADD C132 EQ␣␣ABC* WHERE C1 EQ 2345:
```

If a where-clause is not specified in an update command, you must specify a full trace. If a where-clause is specified, you can specify a partial trace.

Omitting a command terminator at the end of an update command with a where-clause can result in an incorrect number of records being selected for updating. See Chapter 26, "Optimization of Where-Clauses," for details.

Trace Notation

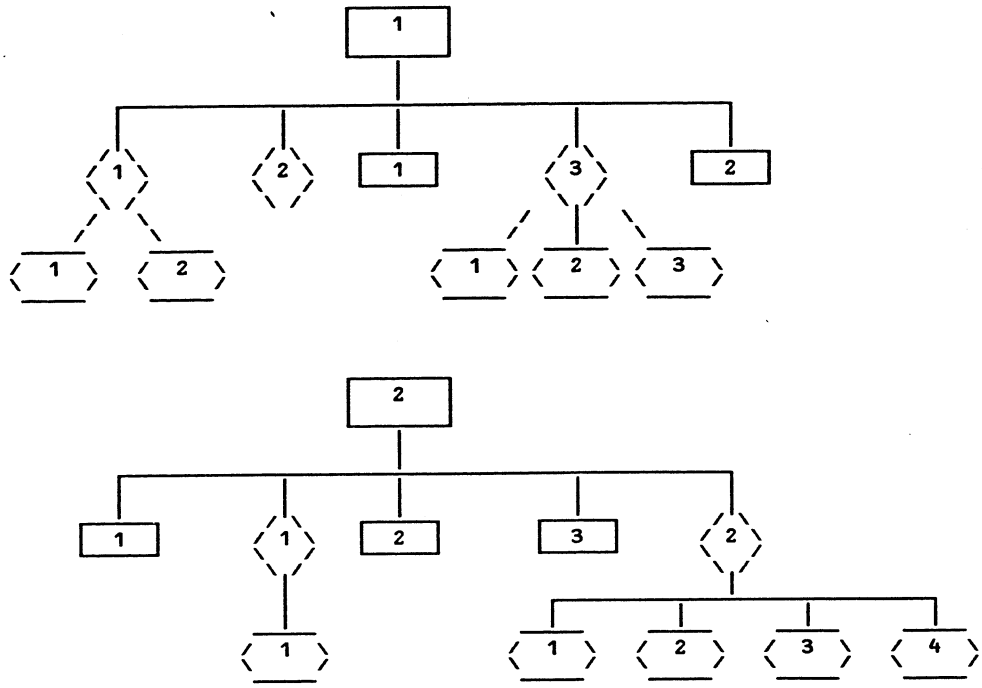
Normalization applies not only to where-clauses but also to update operations, particularly in the area of level adjustment, upward or downward, from the qualified data records. (For information about normalization, see Chapter 25, "Normalizing with the HAS Operator.")

Within update operations, level adjustment is necessary because each data record that is operated upon must belong to the specified schema record. After the necessary normalization is completed for all qualified data records, duplications of selected data records are removed from the resulting list of selected data records. Thus, no update operation will affect a selected data record more than once in a single command.

When you are updating a database, it is sometimes necessary to point to a specific data record in a tree. Pointing identifies the hierarchical linkage for the selected data records by identifying the exact data record at each level. This activity is called *trace notation*.

The following illustration shows the data record numbering logic.

Illus. 20.1 Implied Data Record Numbering Used for Trace Notation



Full trace Trace notation requires some knowledge of the database structure. Trace notation selects a data record by its position as a node in the database without regard to any condition of values. Each integer number in the trace notation signifies the position of a data record within a particular subtree. Integer numbers are separated by the current system separator. The integer number immediately following the component identification refers to the level of the specified data record. The next integer refers to its parent at the next highest level, the next to the parent of the parent, and so on.

For a *full trace*, you must supply a positive integer number for each level, starting with the level of the specified record and ending with level 0. The number zero means the last position on a level.

For each level, the positive number *n* denotes the *n*th position on a level within similar data siblings. For example, the command given below refers to the third occurrence of the SALARY WITHIN POSITION record under the first POSITION WITHIN COMPANY data record of the second logical entry. The ADDITIONAL INFORMATION record occurrences are not counted in determining the position of the affected SALARY WITHIN POSITION data record, nor do other unrelated data records affect the count at level 1.

```
CHANGE C111 *3*1*2 EQ $1900.00*:
```

A where-clause is invalid when you specify a full trace.

The following REMOVE command would remove the fourth logical entry from the database:

```
REMOVE TREE ENTRY *4:
```

In the ASSIGN command below, the software selects the second JOB SKILLS data record in the first logical entry. Its contents are removed and replaced with the values in the value stream.

```
ASSIGN C200 *2*1 EQ 201* CARTOON ART* 202* FAIR* 203*1* END*:
```

In the following INSERT TREE command, a new logical entry is added as the last entry in the database. The logical entry has one data record in which the LAST NAME is SMITH.

```
INSERT TREE ENTRY* 0 EQ 1* 1272* 2* SMITH* 3* GARLAND P.* 6* 397-
   -80-8491* END*:
```

Partial trace A *partial trace* extends upward to any level below level 0. (Trace notation that extends all the way to level 0 is a full trace.) A command with a partial trace always requires a where-clause.

In a command with a partial trace, both the where-clause qualification criteria and the extent of the trace notation determine which data records are affected by the command. Loss of selectivity occurs if the trace notation does not extend to the level of the where-clause.

In general, the where-clause is used for broad selection of data records, and the partial trace serves to isolate the *n*th data record below each qualified data record.

In the next command, the partial trace identifies the first C120 record under the first C110 record under the first C100 record under selected C0 records. Since the where-clause qualifies only one C0 record, the command changes the value of C123 in only one record.

```
CHANGE C123 *1*1*1 EQ 10.00* WHERE C2 EQ REID:
```

However, in the command shown below, the partial trace identifies only the first C120 record under each C110 record that is a descendant of the qualified C0 record. Because the partial trace is not as specific as in the first command, this command changes the value of C123 in all C120 records under REID.

```
CHANGE C123 *1 EQ 10.00* WHERE C2 EQ REID:
```

REPLACING A NULL WITH A VALUE: ADD

The ADD command replaces nulls with values in existing data records. The ADD command

- is restricted to single-level operations
- affects only those data records that are occurrences of a single schema record
- replaces one or more null items with values in selected records
- can include a partial trace in conjunction with a where-clause to pinpoint the position of selected records
- can include a full trace to select one specific data record for updating.

The ADD command adds values only where the specified items have no values. Thus, you can use this command and let the software check the data record contents for the existence of values.

The format you choose for the ADD command is determined by the activity involved: adding a value to a single item or adding values to multiple items in selected records.

Adding a Value for One Item

If a value is to be added to a single item in all of the selected data records, use the formats specifying an item. (The following examples use the EMPLOYEE database.)

```
ADD SECURITY CLEARANCE EQ 106* WHERE C1 EQ 1043:

ADD C106 EQ 02/28/89* WHERE C105 EQ 02/16/88 AND C2 EQ REID:

ADD C124 = (C111 * C122) WHERE C2 EQ WAGGONNER:

ADD C125 = C114 WHERE C2 EQ REID:
```

Adding Values for Multiple Items in a Record

If you want to add new values to several items in selected data records, use the formats specifying a record. When adding multiple values, all values must be for items in the specified record. The software tests each item specified in the value stream independently to see whether it is currently valued in the selected data record before adding the value for that item. Items found to be null have values added; items found to be valued are unaffected. Items in the value stream can be in any order. Items not specified are not affected by the update.

```
ADD EDUCATION EQ 413* 01/07/85* 414* ENGLISH* 415*MARKETING* END*
   WHERE C2 EQ BROOKS:

ADD C0 EQ 11* 40.00* 12* 56.00* END* WHERE C1 EQ 1043:
```

Formats

```
ADD  item  full-trace  EQ  |value *   :
                           |PREVIOUS

ADD  item  full-trace  =  |item                    :
                          |* function *
                          |* function (parameters)
                          |(adhoc-function)
                          |system-function

ADD  item  [partial-trace]  EQ  |value *  where-clause  :
                                |PREVIOUS

ADD  item  [partial-trace]  =  |item                        where-clause:
                               |* function *
                               |* function (parameters)
                               |(adhoc-function)
                               |system-function

ADD  record  full-trace  EQ  |value-stream END *  :
                             |PREVIOUS

ADD  record [partial-trace] EQ |value-stream END * where-clause  :
                               |PREVIOUS
```

item is a schema item name or component number.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

value is a literal value or a system string (*NOW*, *TODAY*, *FTODAY*).

PREVIOUS repeats the value stream or the item value in the preceding update command. When specifying PREVIOUS, the item or record preceding EQ must be the same item or record as in the previous update command. Also, the command referenced by PREVIOUS cannot contain errors.

function is a stored function name or component number.

parameters are the values that replace the numbered parameters in the stored function.

Abbreviations: ADD = AD

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions," for details.

Ad hoc functions always yield a result with three digits to the right of the decimal point. Therefore, if you specify an ad hoc function and the items or constants in the function have four or more digits to the right of the decimal point, the resulting calculation retains only the first three digits to the right of the decimal point, and the value is stored that way in the database.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions," for details.

partial-trace is trace notation in which the last integer designates a position other than level 0.

record is a schema record name or component number.

value-stream is a series of item numbers and values to be added, with each item number followed by its associated value. The system separator must follow each item number and each value. The item numbers do not have to be in DESCRIBE order.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Considerations

- In using the form ADD item$_1$ = item$_2$, the item types must be compatible, that is, both dates, both numeric, or both textual. For numeric items, the second item value assumes the picture designation of the first item type. For example, if C2 equals 15.5 and C1 is INTEGER, then ADD C1=C2 produces the value 15 for C1. However, if C2 is TEXT and C1 is CHARACTER, extraneous blanks are preserved. If the value of the second item is too large for the picture of the first item, an error message appears. Also, if the name of either item includes the equals sign, specify the component number.

  In addition, for the 'ADD item.= item' syntax, if one item is type UNDEFINED, the other item must also be type UNDEFINED. That is, you can add only UNDEFINED values to other UNDEFINED items. You cannot add CHARACTER or TEXT values to UNDEFINED items.

- The item, stored function, system function, or ad hoc function appearing to the right of the equals sign must be in the path of the target record (the record containing the specified item). If a system function includes a by-phrase, the record in the by-phrase must be in the path of the target record, even if the object of the function is not.

- You cannot issue the ADD command for a damaged database.

- You must have U-authority for items and records specified in the action-clause of an ADD command.

- You cannot use Collect File items in the action-clause of an ADD command.

- When you add REAL or DOUBLE values, truncation or padding occurs as needed. No messages appear.

- E-notation is acceptable for all numeric input in SCF operations. The ±n.nnnE±ee format can be used when entering values for INTEGER, DECIMAL, or MONEY items (as well as REAL and DOUBLE) and when specifying constants in functions, where-clauses, and if-clauses. When the software converts a floating point number (E-notation) to fixed format, the result must conform to the picture specified for the INTEGER, DECIMAL, or MONEY item. The E-notation value might not be stored or displayed exactly as entered. For example, the value 10.999877E00 for an INTEGER item becomes 11.

  Input for REAL and DOUBLE items is usually in E-notation, but whole numbers and decimal numbers are acceptable. The output for REAL and DOUBLE values is always standard scientific notation in the form of ±n.nE±ee.

- UNDEFINED input values can contain all 256 EBCDIC characters from X'00' through X'FF'. The values are stored left-justified with trailing binary zero fill if the value is smaller than the picture. Binary zeros are allowed within a value, but an UNDEFINED value containing all binary zeros is considered a missing value (null). You can use display format or HEX notation to input UNDEFINED item values (for example, X'E2E8E2E3C5D4'). For more information about entering UNDEFINED values see **Coding UNDEFINED values** on page 8-17.

REPLACING EITHER A VALUE OR A NULL WITH A VALUE: ASSIGN

The ASSIGN command assigns values within existing data records whether the items are valued or not. The ASSIGN command

- is restricted to single-level operations
- affects only those data records that are occurrences of a single schema record
- assigns one or more new values for several items in selected records
- can include a partial trace in conjunction with a where-clause to pinpoint the position of selected records
- can include a full trace to select a specific data record for updating.

The ASSIGN command alters the contents of selected data records by replacing values or nulls with the new values specified. Unlike the ADD or CHANGE commands, which are conditional (depending upon the status of the values for the items), ASSIGN is unconditional in that it always assigns the new value if the item does not already contain the new value.

An ASSIGN command affects only selected data records, that is, those occurrences of the target record that are related to the qualified data records. The format you choose for the ASSIGN command is determined by the activity involved: assigning a value to a single item or assigning new values to multiple items.

Assigning a Value for One Item in a Record

If a value is to be entered for a single item in all selected data records, you can use the formats that specify an item. The software checks each selected data record. If the item has a value equal to the new specified value, no action takes place; otherwise, any other value, if it exists, is removed and replaced by the new value.

```
ASSIGN OFFICE-EXTENSION EQ 410 XT369* WHERE C1 EQ 1120:

ASSIGN C414 EQ PHYS ED* WHERE C2 EQ SCHOLL:
```

Assigning Values for Several Items in a Record

If new values are to be assigned to several items in selected data records, you can use the formats that specify a record. When assigning multiple values, all values must be occurrences of items in the specified record. The software checks each item in the selected data record being acted upon.

If the current value for an item is the same as the new value specified for that item in the value stream, the item is not affected. Otherwise, any other value, if it exists, is removed and replaced by the new value specified.

Data for items not specified in the value stream are removed from the database. That is, if no value is specified for an item in the value stream, the item becomes null. Therefore, when the formats specifying à record are used and any of the original values are to be retained, the value stream must include the original value. Otherwise, those data will be lost in the removal.

Items in the value stream can be in any order.

```
ASSIGN JOB SKILLS EQ 201* RUSSIAN* 202* EXCELLENT*203* 15* END*
   WHERE C2 EQ BOWMAN:

ASSIGN C200 EQ 201* PASCAL* 202* EXCELLENT* 203* 3* END*
   WHERE C1 EQ 1050:
```

Formats

```
ASSIGN  item  full-trace  EQ  |value *    :
                              |PREVIOUS

ASSIGN  item  full-trace  =  |item                        :
                             |* function *
                             |* function (parameters)
                             |(adhoc-function)
                             |system-function

ASSIGN  item  [partial-trace]  EQ  |value *   where-clause :
                                   |PREVIOUS

ASSIGN  item  [partial-trace] =  |item                 where-clause   :
                                 |* function *
                                 |* function (parameters)
                                 |(adhoc-function)
                                 |system-function

ASSIGN  record  full-trace  EQ  |value-stream END *  :
                                |PREVIOUS

ASSIGN  record [partial-trace]  EQ  |value-stream END * where-clause:
                                    |PREVIOUS
```

item is a schema item name or component number.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

value is a literal value or a system string (*NOW*, *TODAY*, *FTODAY*).

PREVIOUS repeats the value stream or the item value in the preceding update command. When specifying PREVIOUS, the item or record preceding EQ must be the same item or record as in the previous update command. Also, the command referenced by PREVIOUS cannot contain errors.

function is a stored function name or component number.

parameters are the values that replace the numbered parameters in the stored function.

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions," for details.

Ad hoc functions always yield a result with three digits to the right of the decimal point. Therefore, if you specify an ad hoc function and the items or constants in the function have four or more digits to the right of the decimal point, the resulting calculation retains only the first three digits to the right of the decimal point, and the value is stored that way in the database.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions," for details.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

record is a schema record name or component number.

value-stream is a series of item numbers and values to be assigned, with each item number followed by its associated value. The system separator must follow each item number and each value. The item numbers do not have to be in DESCRIBE order.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Considerations

- In using the form ASSIGN item$_1$ = item$_2$ the item types must be compatible, that is, both dates, both numeric, or both textual. For numeric items, the second item value assumes the picture designation of the first item type. For example, if C2 equals 15.5 and C1 is INTEGER, then ASSIGN C1=C2 produces the value 15 for C1. However, if C2 is TEXT and C1 is CHARACTER, extraneous blanks are preserved. Also, if the value of the second item is too large for the picture of the first item, an error message appears. If the item name for either includes the equals sign, use the component number for the item.

  In addition, for the 'ASSIGN item = item' syntax, if one item is type UNDEFINED, the other item must also be type UNDEFINED. That is, you can assign only UNDEFINED values to other UNDEFINED items. You cannot assign CHARACTER or TEXT values to UNDEFINED items.

- The item, stored function, or ad hoc function appearing to the right of the equals sign must be in the path of the target record (the record containing the specified item). If a system function includes a by-phrase, the record specified in the by-phrase must be in the path of the target record, even if the object of the function is not.

- You must have U-authority for items and records specified in the action-clause of an ASSIGN command.

- You cannot use Collect File items in the action-clause of an ASSIGN command.

- You cannot issue the ASSIGN command for a damaged database.

- When you assign REAL or DOUBLE values, truncation or padding occurs as needed. No messages appear.

- E-notation is acceptable for all numeric input in SCF operations. The ±n.nnnE±ee format can be used when entering values for INTEGER, DECIMAL, or MONEY items (as well as REAL and DOUBLE) and when specifying constants in functions, where-clauses, and if-clauses. When the software converts a floating point number (E-notation) to fixed format, the result must conform to the picture specified for the INTEGER, DECIMAL, or MONEY item. The E-notation value might not be stored or displayed exactly as entered. For example, the value 10.999877E00 for an INTEGER item becomes 11.

  Input for REAL and DOUBLE items is usually in E-notation, but whole numbers and decimal numbers are acceptable. The output for REAL and DOUBLE values is always standard scientific notation in the form of ±n.nE±ee.

- UNDEFINED input values can contain all 256 EBCDIC characters from X'00' through X'FF'. The values are stored left-justified with trailing binary zero fill if the value is smaller than the picture. Binary zeros are allowed within a value, but an UNDEFINED value containing all binary zeros is considered a missing value (null). You can use display format or HEX notation to input UNDEFINED item values (for example, X'E2E8E2E3C5D4'). For more information about entering UNDEFINED values see **Coding UNDEFINED values** on page 8-17.

REPLACING A VALUE WITH A VALUE: CHANGE

The CHANGE command replaces values with new values. The CHANGE command

- is restricted to a single-level operation
- affects only those data records that are occurrences of a single schema record
- replaces one or more values with corresponding new values in selected records
- can include a partial trace in conjunction with a where-clause to pinpoint the position of selected records
- can include a full trace to select one specific data record for updating.

The CHANGE command changes values only where the specified items have values. For each selected data record, the software looks at the status of each item specified in the command. If the item has a value, it is changed to the new value specified in the value stream. If the item has no value, no action is taken.

The CHANGE command changes existing values; it never adds new values. The format you use for the CHANGE command is determined by the activity involved: changing a value for a single item or changing values for multiple items in selected records.

Changing a Value for One Item in a Record

If a value is to be changed for a single item in selected data records, you can use the formats that specify an item, as shown in the following examples.

```
CHANGE PAY SCHEDULE EQ MONTH* WHERE C112 EQ MONTHLY:

CHANGE C11 EQ 80.00* WHERE C1 EQ 1120:

CHANGE C131 = (C131/10) WHERE C1 EQ 1043:
```

Changing Values for Several Items in a Record

If the values for several items are to be changed in selected data records, you can use the formats that specify a record. When changing multiple values, all values must be occurrences of items in the specified record. The software independently tests each item specified in the value stream to see whether it is currently null in the selected data record before changing the value for that item.

Items in the value stream can be in any order.

Items found to be valued are changed; items found to be null are not changed. Existing values for items not specified in the value stream remain unaffected.

```
CHANGE JOB SKILLS EQ 202* GOOD* 203* 3* END* WHERE C201 EQ
   GRAPHICS AND C1 EQ 1043:

CHANGE C0 EQ 14* 1302 LAZY LANE* 16* 78752* END* WHERE C1 EQ 1120:
```

Formats

```
CHANGE item  full-trace  EQ  |value *    :
                             |PREVIOUS

CHANGE item  full-trace  =  |item                        :
                            |* function *
                            |* function (parameters)
                            |(adhoc-function)
                            |system-function

CHANGE item  [partial-trace]  EQ  |value *   where-clause  :
                                  |PREVIOUS

CHANGE item  [partial-trace]  =  |item                 where-clause :
                                 |* function *
                                 |* function (parameters)
                                 |(adhoc-function)
                                 |system-function

CHANGE record  full-trace  EQ  |value-stream END *  :
                               |PREVIOUS

CHANGE record [partial-trace]  EQ  |value-stream END * where-clause:
                                   |PREVIOUS
```

item is a schema item name or component number.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

value is a literal value or a system string (*NOW*, *TODAY*, *FTODAY*).

PREVIOUS repeats the value stream or the item value in the preceding update command. When specifying PREVIOUS, the item or record preceding EQ must be the same item or record as in the previous update command. Also, the command referenced by PREVIOUS cannot contain errors.

function is a stored function name or component number.

parameters are the values that replace the numbered parameters in the stored function.

Abbreviations: CHANGE = CH

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions," for details.

Ad hoc functions always yield a result with three digits to the right of the decimal point. Therefore, if you specify an ad hoc function and the items or constants in the function have four or more digits to the right of the decimal point, the resulting calculation retains only the first three digits to the right of the decimal point. The value is stored that way in the database.

system-function allows you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM. See Chapter 34, "Obtaining Arithmetic Statistics: System Functions," for details.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

record is a schema record name or component number.

value-stream is a series of item numbers, each followed by the associated value to be changed. The system separator must follow each item number and each value. The item numbers do not have to be in DESCRIBE order.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Considerations

- For CHANGE $item_1$ = $item_2$ the item types must be compatible, that is, both dates, both numeric, or both textual. For numeric items, the first item value assumes the picture designation of the second item type. For example, if C2 equals 15.5 and C1 is INTEGER, then CHANGE C1=C2 produces 15 for C1. If C2 is TEXT and C1 is CHARACTER, extraneous blanks are kept. If the value of the second item is too large for the picture of the first item, an error message appears. Also, if the item names include the equals sign, the component numbers should be specified.

  In addition, for the 'CHANGE item = item' syntax, if one item is type UNDEFINED, the other item must also be type UNDEFINED. That is, you can change only UNDEFINED values to other UNDEFINED items. You cannot change CHARACTER or TEXT values to UNDEFINED items.

- The item, stored function, system function, or ad hoc function appearing to the right of the equals sign must be in the path of the target record (the record containing the specified item). If a system function includes a by-phrase, the record in the by-phrase must be in the path of the target record, even if the object of the function is not.

- You cannot issue the CHANGE command for a damaged database.

- You must have U-authority for items or records specified in the action-clause of a CHANGE command.

- You cannot use Collect File items in the action-clause of a CHANGE command.

- When you change REAL or DOUBLE values, truncation or padding occurs as needed. No messages appear.

- E-notation is acceptable for all numeric input in SCF operations. The ±n.nnnE±ee format can be used when entering values for INTEGER, DECIMAL, or MONEY items (as well as REAL and DOUBLE) and when specifying constants in functions, where-clauses, and if-clauses. When the software converts a floating point number (E-notation) to fixed format, the result must conform to the picture specified for the INTEGER, DECIMAL, or MONEY item. The E-notation value might not be stored or displayed exactly as entered. For example, the value 10.999877E00 for an INTEGER item becomes 11.

  Input for REAL and DOUBLE items is usually in E-notation, but whole numbers and decimal numbers are acceptable. The output for REAL and DOUBLE values is always standard scientific notation in the form of ±n.nE±ee.

- UNDEFINED input values can contain all 256 EBCDIC characters from X'00' through X'FF'. The values are stored left-justified with trailing binary zero fill if the value is smaller than the picture. Binary zeros are allowed within a value, but an UNDEFINED value containing all binary zeros is considered a missing value (null). You can use display format or HEX notation to input UNDEFINED item values (for example, X'E2E8E2E3C5D4'). For more information about entering UNDEFINED values see **Coding UNDEFINED values** on page 8-17.

REPLACING A VALUE WITH A NULL: REMOVE

The REMOVE command replaces values with nulls in selected data records. The REMOVE command

- is restricted to single-level operations
- affects only those data records that are occurrences of a single schema record
- removes values for one or all items in selected records
- can include a partial trace in conjunction with a where-clause to pinpoint the position of selected records
- can include a full trace to select a specific data record for updating.

The REMOVE command removes the value of either a single item or all items in each selected data record.

Removing a Value for One Item in a Record

If you use the formats that specify an item, the value is removed for each selected data record when the item has a value. If the item has no value, no action occurs.

```
REMOVE CURRENT DEDUCTION WHERE C113 EQ 03/01/90 AND C1 EQ 1120:

REMOVE C302 WHERE C1 EQ 1043:
```

Removing Values for Multiple Items in a Record

If you use the formats that specify a record, every value is removed from every selected data record. The data records themselves are never removed by this command. The record remains in the database as a null record. (Only REMOVE TREE removes records.)

```
REMOVE ENTRY WHERE C1 FAILS:
```

Formats

```
REMOVE  item  full-trace  :

REMOVE  item  [partial-trace]  where-clause  :

REMOVE  record  full-trace  :

REMOVE  record  [partial-trace]  where-clause  :
```

item is a schema item name or component number.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

record is a schema record name or component number.

Considerations

- You cannot issue the REMOVE command for a damaged database.
- You must have U-authority for items and records specified in the action-clause of a REMOVE command.
- You cannot use Collect File items in the action-clause of a REMOVE command.

Chapter 21

Updating Data Trees

INTRODUCTION 21-2
Output From Update Commands 21-2

REPLACING ONE DATA TREE WITH ANOTHER: ASSIGN TREE 21-4
Formats 21-5
Considerations 21-6
Examples 21-6

COPYING DATA TREES: COPY TREE 21-10
Formats 21-10

ADDING NEW DATA TREES: INSERT TREE 21-15
Formats 21-15
Inserting a New Subtree before or after Selected Records 21-16
Inserting a New Subtree Using Partial Trace Notation and a Where-Clause 21-16
Inserting a New Subtree Using Full Trace and No Where-Clause 21-17
Considerations 21-18
Examples Specifying BEFORE/AFTER 21-18
Examples Specifying Partial Trace with a Where-Clause 21-25
Examples Specifying Full Trace and No Where-Clause 21-34

MOVING DATA TREES: MOVE TREE 21-37
Formats 21-38
Considerations 21-39
Examples 21-40

REMOVING DATA TREES: REMOVE TREE 21-42
Formats 21-43
Considerations 21-43
Examples 21-43

INTRODUCTION

The commands discussed in this chapter modify the structure of a database either by inserting new data records or by removing, moving, or copying data trees. There are five commands to modify database structure:

- The **ASSIGN TREE** command replaces an existing data tree with a new data tree.
- The **COPY TREE** command copies a data record and its descendants from one position in the database to another.
- The **INSERT TREE** command adds new records, new data trees, or new logical entries to a database.
- The **MOVE TREE** command moves a data tree from one position in the database to another.
- The **REMOVE TREE** command removes data trees (including logical entries) from a database. That is, it removes a data record and all its descendants.

If only values within existing data records are to be modified, use the commands discussed in Chapter 20, "Updating Data Records."

The database cycle number is incremented at the successful completion of each update command. This number is displayed in the output of the DESCRIBE command.

Output From Update Commands

No output results from update commands. However, three types of messages can occur as a result of an update command:

- an error message
- a message indicating the number of selected data records
- the -373- DATA BASE UNALTERED- message.

To verify the results of update commands, you can issue appropriate retrieval commands following the updates.

The database is marked internally if any update process completes abnormally, as in the occurrence of fatal errors, time limit job abends, operator drops, system drops, and so forth. (Syntax errors do not cause damage to a database.) Subsequently, if you try to update a database damaged from a previous job, the software issues the following message:

```
-810- WARNING- DATA BASE DAMAGED.  NO UPDATES PERMITTED -
```

This message appears in the Message File of every job using a database damaged by a previous job abend. The message is also printed if you attempt to use an update command. However, you can still save, restore, describe, and retrieve from the database.

The concepts of qualified records, selected records, and normalization also apply while the software processes updates, particularly in the area of level adjustment, upward or downward, from the qualified data records. (For information about normalization, see Chapter 25, "Normalizing with the HAS Operator.")

Within update operations, level adjustment is necessary because the top of each selected tree must belong to the specified schema record, although selection of an ancestor is allowed for some INSERT TREE commands. After all the necessary normalization has been done for all qualified data records, duplications of selected data records are removed from the resulting list of selected data records. Thus, with the exception of INSERT TREE operations, updates never affect a selected data record more than once in a single command. Selecting records above the level of insertion allows multiple insertions below the selected records.

REPLACING ONE DATA TREE WITH ANOTHER: ASSIGN TREE

The ASSIGN TREE command replaces existing data trees (including logical entries) with new data trees. The ASSIGN TREE command

- affects only existing data trees
- can include a partial trace in conjunction with a where-clause to pinpoint the location of selected records
- can include a full trace to select a specific data tree for updating
- cannot be used to insert new trees (see INSERT TREE).

The ASSIGN TREE command always specifies a target record. The where-clause (if present) and trace notation (if present) select records that are occurrences of the target record. When the command is executed, these selected records and their descendants are removed from the database and replaced by the new records specified in the value stream.

The value stream contains data for the target record and its descendant records in logical order. The record number of the target record is not included in the value stream.

If you are replacing an entire logical entry, the target record must be C0 and the value stream must be identical to a loader stream for the entry.

Formats

```
ASSIGN TREE record  full-trace  EQ  |value-stream END* :
                                    |PREVIOUS
```

```
ASSIGN TREE record [partial-trace] EQ |value-stream END* where-clause:
                                      |PREVIOUS
```

record is a schema record name or component number; the target record.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

value-stream is a series of component numbers and associated item values in the same format as was used for the loader stream. The record number to the left of the EQ operator is not included in the value stream. See Chapter 8, "Using the LOAD Command," for the format of a loader stream.

If any of the original data are to be retained, they must be included as part of the value stream. Otherwise, they will be removed from the database.

When a value stream is not included, the selected record and its descendants are replaced by an empty record without descendants.

PREVIOUS repeats the value stream or the item value in the preceding update command. When specifying PREVIOUS, the schema record preceding EQ must be the same schema record as in the previous update command. Also, the command referenced by PREVIOUS cannot contain errors.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Abbreviations: ASSIGN TREE = AT

Considerations

- You cannot issue the ASSIGN TREE command for a damaged database.
- The space in the Data Table that was occupied by the replaced records becomes part of reusable space. In a Multi-User environment, the space is not immediately reused when more than one user is accessing the database. The new top record always occupies the same space as the old top record.
- The software alters the database when an ASSIGN TREE is issued, even if the values specified in the value stream are the same as the values already stored in the database; that is, the message -373- DATA BASE UNALTERED - is not issued.
- You must have U-authority for the items and records specified in the action-clause of an ASSIGN TREE command.
- You cannot use Collect File items in the action-clause of an ASSIGN TREE command.
- When you assign REAL or DOUBLE values, truncation or padding occurs as needed. No messages appear.
- E-notation is acceptable for all numeric input in SCF operations. You can use the ±n.nnnE±ee format when entering values for INTEGER, DECIMAL, or MONEY items (as well as REAL and DOUBLE) and when specifying constants in functions, where-clauses, and if-clauses. When the software converts a floating point number (E-notation) to fixed format, the result must conform to the picture specified for the INTEGER, DECIMAL, or MONEY item. The E-notation value might not be stored or displayed exactly as entered. For example, the value 10.999877E00 for an INTEGER item becomes 11.

  Input for REAL and DOUBLE items is usually in E-notation, but whole numbers and decimal numbers are acceptable. The output for REAL and DOUBLE values is always standard scientific notation in the form of ±n.nE±ee.
- UNDEFINED input values can contain all 256 EBCDIC characters from X'00' through X'FF'. The values are stored left-justified with trailing binary zero fill if the value is smaller than the picture. Binary zeros are allowed within a value, but an UNDEFINED value containing all binary zeros is considered a missing value (null).

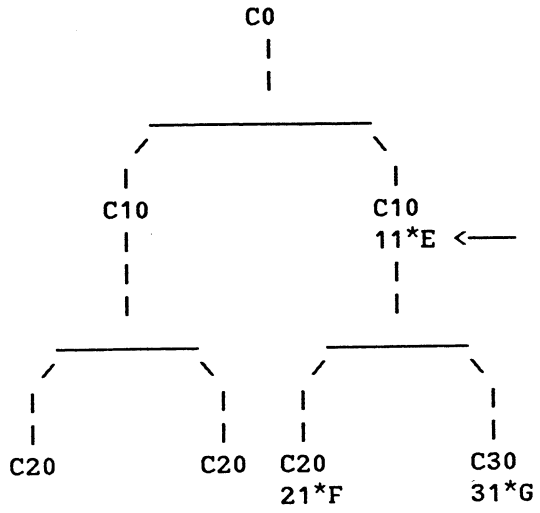
  You can use display format or HEX notation to input UNDEFINED item values (for example, X'E2E8E2E3C5D4'). For more information about entering UNDEFINED values see **Coding UNDEFINED values** on page 8-17.

Examples

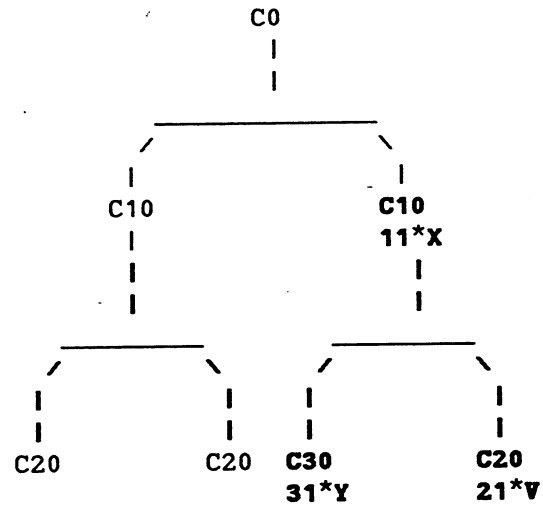
Examples 1 through 3 show the effect of ASSIGN TREE commands. In each example, two data trees are shown. The first shows the data tree before the command is executed; arrows point to the selected records. The second tree shows the data tree with the records that have been changed by the command. These records are bolded.

Example 1

This example shows an ASSIGN TREE command with a where-clause. The value stream includes values for C10, C20, and C30 records.

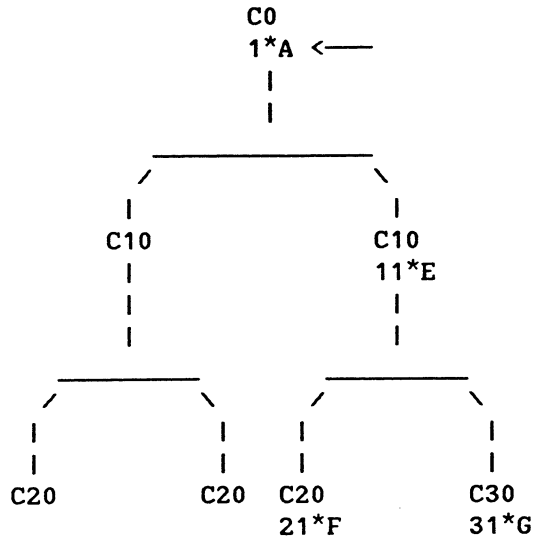



```
ASSIGN TREE C10 EQ 11*X*30*31*Y*20*21*V*END* WHERE C21 EQ F:
```

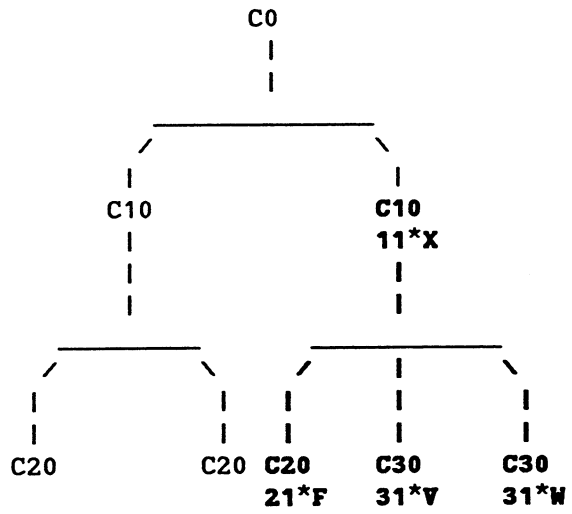


Example 2

This example shows an ASSIGN TREE command with a partial trace and a where-clause. The value stream includes values for C10, C20, and C30 records. The partial trace establishes selection at level 1, and it replaces the second record (and its descendants) under the selected parent with the specified subtree. Notice that the value for the C20 record is retained because it is included in the value stream.

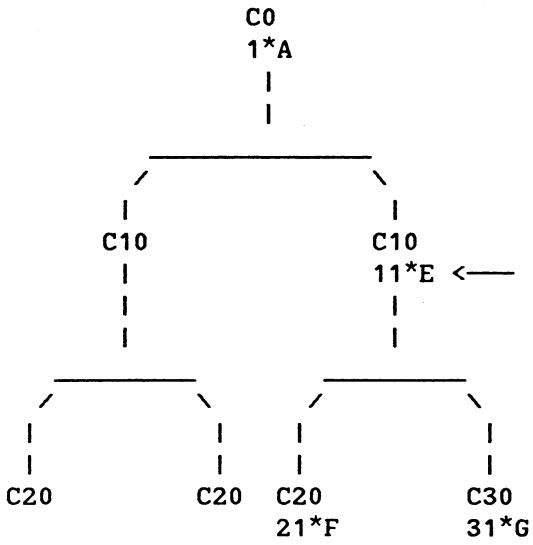



```
ASSIGN TREE C10 *2 EQ 11*X*20*21*F*30*31*V*30*31*W*END* WHERE C1 EQ A:
```

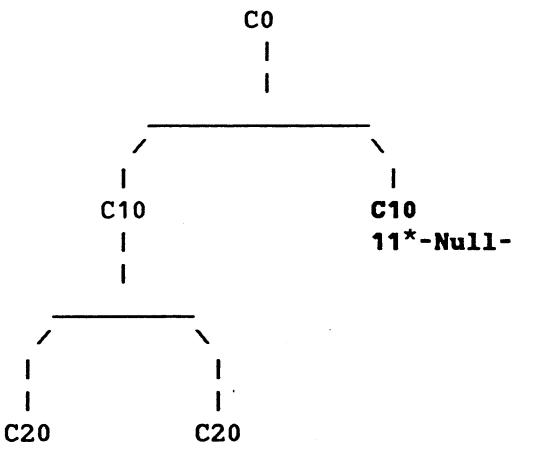


Example 3

This example shows an ASSIGN TREE command with a where-clause and no value stream. The C20 and C30 records under the C10 record with C11 equals E are removed. The values in the C10 record are removed, although the record itself remains in the database.




```
ASSIGN TREE C10 EQ END* WHERE C11 EQ E:
```



COPYING DATA TREES: COPY TREE

The COPY TREE command copies a data record and its descendants from one position in the database to another. The data tree to be copied can be an entire logical entry or any subtree in a logical entry. Copying a data tree causes changes in the Hierarchical Table and the Data Table. If the data tree copied contains key values, the index tables are also changed. The COPY TREE command is not available in queue mode, load mode, or PLEX programs.

The copy operation requires a one-to-one relationship. That is, a data tree is copied from its old position to a single new point of attachment. It cannot be copied to multiple new positions.

Formats

```
COPY TREE record |where-clause                    TO  |before-clause  :
                 |partial-trace where-clause          |after-clause
                 |full-trace                          |partial-trace where-clause
                                                      |full-trace
```

record is the name or C-number of the schema record at the top node of the tree to be copied.

where-clause is any QUEST where-clause, which is used to select specific data records according to your selection criteria. The where-clause begins with the keyword WHERE.

partial-trace is a trace in which the rightmost integer designates a position other than level 0.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost digit designates the level 0 position; that is, a trace in this syntax means a full trace. Zero means last.

before-clause is the same as the where-clause except that the keyword BEFORE replaces the keyword WHERE.

after-clause is the same as the where-clause except that the keyword AFTER replaces the keyword WHERE.

Abbreviations:
AFTER = AF
BEFORE = BE
COPY TREE = CT

COPY TREE has two where-clauses (or partial/full traces). The first where-clause (or trace) selects the top node (record) of the tree or subtree to be copied. The second where-clause (or trace) selects the point at which the new tree is to be inserted. That is, the first where-clause establishes a "from" position, and the second where-clause establishes a "to" position. The selection criteria must qualify only one record for the "from" position and only one record for the "to" position, or an error message is issued.

The rules for the INSERT TREE command establish the parent or the sibling point of attachment except that the "to" position is limited to a single selected record. If you establish the "to" position with a full or partial trace, the software inserts the selected tree or subtree before the selected "to" position.

It is possible to force a COPY TREE to execute if you specify a LIMIT 1, TRUNCATE command. However, the single record selected for the "from" and the "to" position will be the first ones in physical order. Therefore, you should avoid using the TRUNCATE option because there is no guarantee which tree will be copied or where the point of attachment will be.

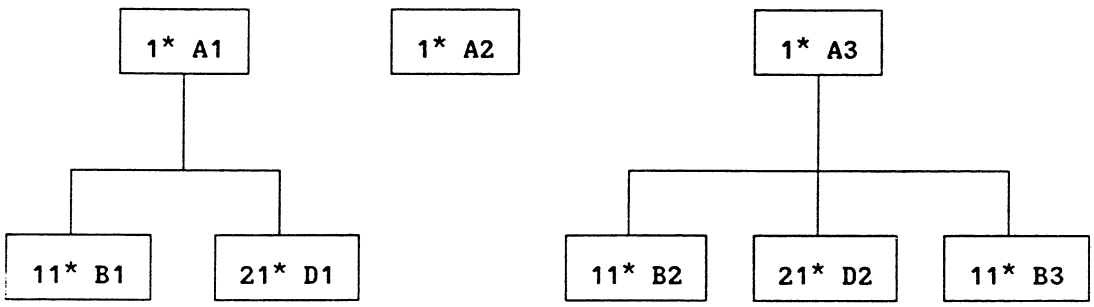
You can use SAME in either or both of the where-clauses. The DYNAMIC and STATIC settings are honored in the same way as any other consecutive where-clauses. Therefore, the "from" where-clause can affect the "to" where-clause, depending on which setting is used for SAME. After a COPY TREE completes processing, the SAME file contains the results of the specified "to" where-clause if SAME is set to DYNAMIC.

You must have U-authority for the components in the action-clause of the COPY TREE command. Also, you cannot copy a tree if the database is damaged.

If Security by Entry is in effect for the database, you can use a secondary password to copy a subtree from one logical entry to another, provided you know the entry key values for both logical entries. The last where-clause used in setting up the "to" position determines the current entry key after the COPY TREE command completes processing.

Note: Although you cannot specify format options with COPY TREE, this command honors the TREE/RECORD format option currently in effect. That is, if RECORD is specified in a previous command, COPY TREE duplicates only the top node of the selected tree.

Examples 1 through 5 use the three hypothetical logical entries shown below. The level 0 record has one item (C1). Record C10 and record C20 are at level 1. Each record has one item, C11 and C21, respectively. In the diagram, 1* A1 means C1 has a value of A1 in that level 0 record. Similarly, 11* B1 represents a C10 record having C11 equal to B1 at level 1.



Because you can use either a full trace or a partial trace for either where-clause in the COPY TREE command, you can specify the "from" position and the "to" position in many ways. The following examples illustrate various COPY TREE commands that accomplish the same results.

Example 1

This example shows copying an entire logical entry from one place to another. The "from" record is the level 0 record containing C1 equal to A1. Any command shown here copies that logical entry and places the copy between the second and third logical entries, that is, between the records containing A2 and A3.

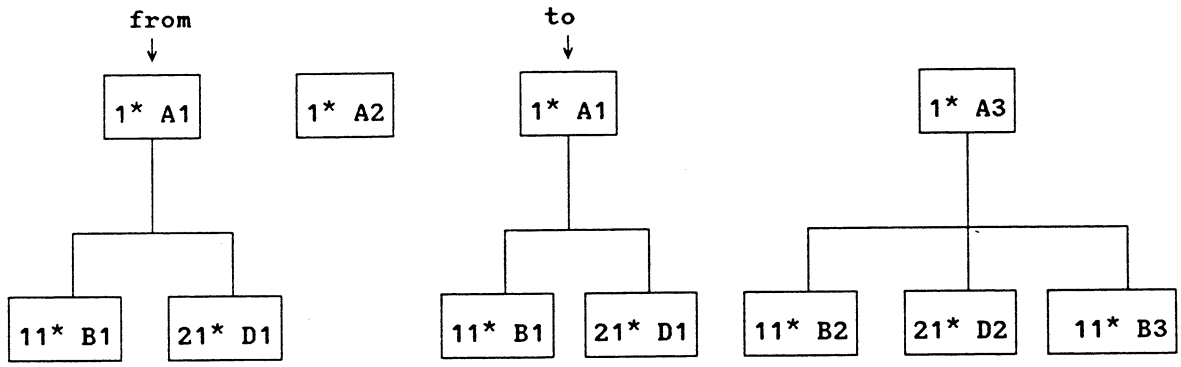
COPY TREE ENTRY WHERE C1 EQ A1 TO AFTER C1 EQ A2:

COPY TREE C0 *1 TO BEFORE C11 EQ B2:

COPY TREE C0 WHERE C21 EQ D1 TO BEFORE C1 EQ A3:

COPY TREE ENTRY WH C1 EQ A1 TO *3:

CT C0 *1 TO *3:



Example 2

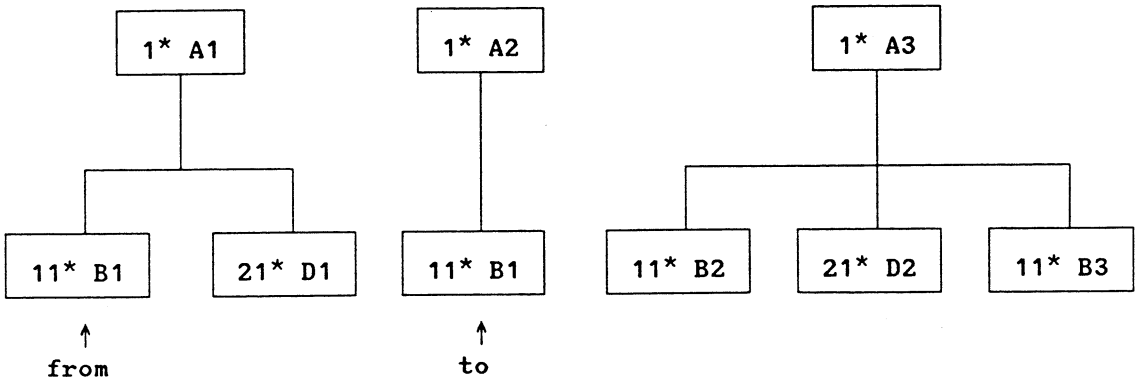
To copy the level 1 record containing B1 to a position under the level 0 record A2, you could use any of the following equivalent commands:

COPY TREE C10 WHERE C11 EQ B1 TO *1 WHERE C1 EQ A2:

COPY TREE C10 WHERE C11 EQ B1 TO *0 WHERE C1 EQ A2:

COPY TREE C10 *1*1 TO *0*2:

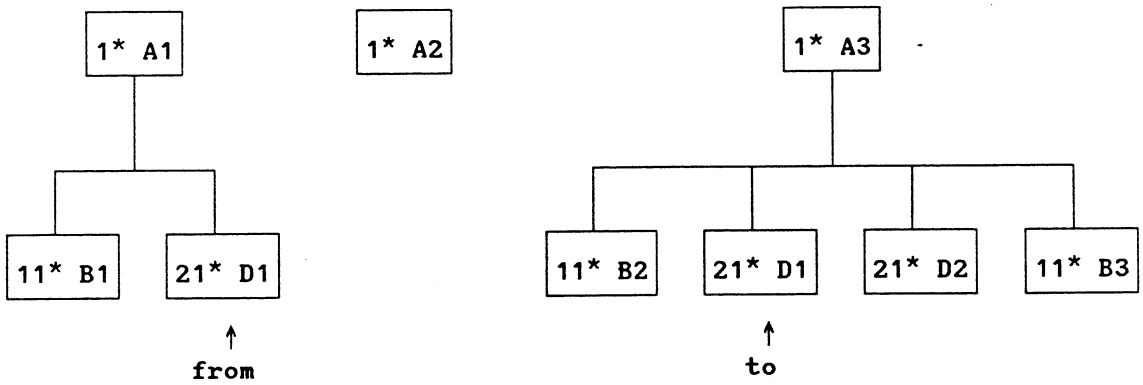
COPY TREE C10 *1*1 TO *1*2:



Example 3

Here is one way to copy the D1 record to a position between the B2 and D2 records.

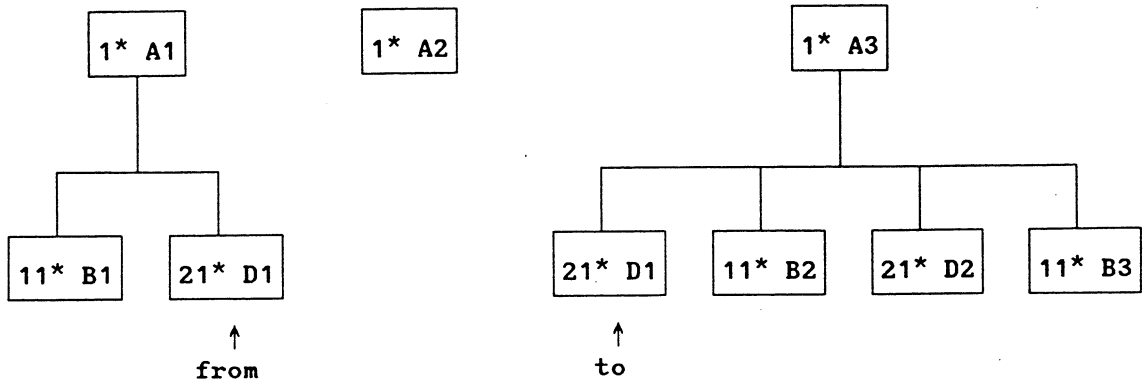
COPY TREE C20 WHERE C21 EQ D1 TO BEFORE C21 EQ D2:



Example 4

Here is one way to copy the D1 record to a position before the B2 record.

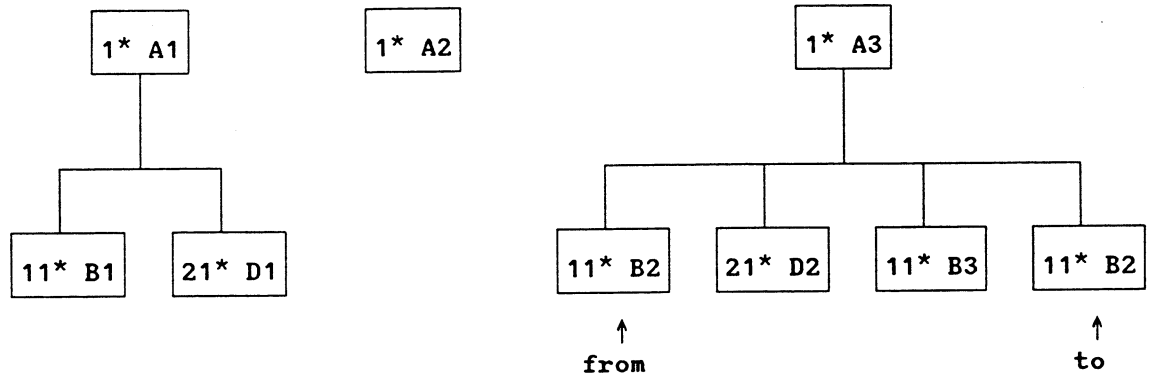
COPY TREE C20 WH C21 EQ D1 TO *1 WHERE C1 EQ A3:



Example 5

Here is one way to copy the first data record under A3 to the end of its level 1 sibling chain under A3, that is, copy the first level 1 record as the last one. Notice the convenient use of SAME.

COPY TREE C10 *1 WHERE C1 EQ A3 TO *0 WHERE SAME:



ADDING NEW DATA TREES: INSERT TREE

The INSERT TREE command adds new data trees where trees do not exist. This command

- adds new records to the database
- adds new subtrees to the database
- adds logical entries to the database
- allows you to pinpoint the exact location of the new subtree.

There are three formats for the INSERT TREE command:

- inserting a new subtree before or after selected records
- inserting a new subtree using partial trace and a where-clause
- inserting a new subtree using full trace and no where-clause.

Formats

```
INSERT TREE  record  EQ  |value-stream END*  |before-clause  :
                         |PREVIOUS           |after-clause
```

```
INSERT TREE  record  full-trace  EQ  |value-stream END*  :
                                     |PREVIOUS
```

```
INSERT TREE  record  partial-trace  EQ |value-stream END* where-clause :
                                       |PREVIOUS
```

record is a schema record name or component number.

value-stream is a series of component numbers and associated item values in the same format as the loader stream. The record number to the left of the EQ operator is not included in the value stream. See Chapter 8, "Using the LOAD Command," for the format of a loader stream.

When the value stream is not included, an empty record is added to the database. Also, if a record number is followed in the value stream immediately by the system separator and another record number, an empty record is inserted for the first record.

PREVIOUS repeats the value stream in the preceding update command. When specifying PREVIOUS, the schema record preceding EQ must be the same schema record as in the previous update command. Also, the command referenced by PREVIOUS cannot contain errors.

Abbreviations:
AFTER = AF
BEFORE = BE
INSERT TREE = IT

before-clause is the same as a where-clause, except that the keyword BEFORE replaces the keyword WHERE. See Chapter 24, "The Where-Clause," for details. You must use a before-clause or an after-clause if you do not use trace notation.

after-clause is the same as a where-clause, except that the keyword AFTER replaces the keyword WHERE. See Chapter 24, "The Where-Clause," for details. You must use a before-clause or an after-clause if you do not use trace notation.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

If you include a full trace, you cannot include a where-clause.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details. If you specify a where-clause, you must also include a partial trace.

Inserting a New Subtree before or after Selected Records

When the INSERT TREE command includes a before-clause (or an after-clause), it inserts the top record of the specified subtree immediately before (or after) records similar to the specified record. Selection is of records similar to the specified record; therefore, the command does not insert a subtree where similar records do not occur.

The before-clause (or the after-clause) qualifies records that determine the selected records serving as reference points for positioning of the new specified subtree. That is, if the command includes a before-clause (or an after-clause), the record at the top of the new subtree is placed immediately before (or after) each selected record. The command is processed only if the specified record is related to the records to be qualified by the before-clause (or the after-clause).

Inserting a New Subtree Using Partial Trace Notation and a Where-Clause

An INSERT TREE command with a partial trace and a where-clause inserts the top record of the specified subtree in a specific position under selected records. The leftmost digit of the partial trace determines the position of the record under its parent and among its similar siblings.

The number of numbers in the trace notation determines the selection level. Selection is at the level of the top record of the specified schema record minus the number of digits in the trace notation. For example, if the specified record is at level 4 and there are 3 numbers in the trace notation, the selection is at level 1.

The schema record at the selection level must be related to the records qualified by the where-clause. The records qualified by the where-clause determine initial record selection, that is, those records selected as parents or ancestors of the new top record.

The software must also select records to serve as reference points for insertion. These records are selected according to initial qualification/selection via the where-clause, plus trace notation that selects according to the record's position beneath the selected records. For instance, if only one number is specified in the trace notation, the parents of the new records are selected according to the qualification criteria specified in the where-clause.

At the other extreme, if the numbers in the trace notation specify the position of records through level 1, the new record is placed in the logical position indicated by the trace notation and under those C0 records that are selected in the paths of the qualified records.

When the partial trace begins with a 1 (insert in first position) or 0 (insert in last position), a parent of the specified record can be selected as the point of insertion. When a parent record has no descendants similar to the specified record, it is selected as a point of insertion only if the partial trace begins with a 1 or 0. If the partial trace begins with other than 1 or 0, records that are similar to the specified record are chosen as points of insertion. In general, the position specified by the leftmost number in the trace notation must take into account the positions of existing similar records. For example, if 4 is specified in trace notation, the new record is inserted only under those parents having at least three records similar to the new record.

In summary, the following table illustrates what record is selected as a point of insertion and where the new record is inserted, provided that the initial number in the trace notation is 0, 1, or 2.

	1st Trace Number	Record Selected at Point of Insertion	Position Where New Record is Inserted
If similar sibling records exists	0	Last similar sibling	After last similar sibling record
	1	Parent	First under parent record
	2	First similar sibling	After first similar sibling record
If similar sibling records do not exist	0	Parent	First under parent record
	1	Parent	First under parent record
	2	None	None

Inserting a New Subtree Using Full Trace and No Where-Clause

An INSERT TREE command with a full trace inserts the top record of the specified subtree in the position specified by the trace notation. The leftmost digit of the trace notation determines the position of the new record under its parent and among its similar siblings. Each subsequent digit in the trace notation determines the position of each of the ancestors of the record up to level 0. That is, the full trace specifies the unique position of the top record in the database.

Like partial trace, if a parent record has no descendants similar to the specified record, insertion will take place only if the partial trace begins with a 1 (insert in first position) or a 0 (insert in last position). In general, the position specified by the leftmost digit of the trace notation must take into account the positions of existing similar records. For example, if 4 is specified in the trace notation, the new record is inserted only if the parent has at least three records similar to the new record.

Considerations

- You cannot issue the INSERT TREE command for a damaged database.
- You must have U-authority for items and records specified in the action-clause of an INSERT TREE command.
- You cannot use Collect File items in the action-clause of an INSERT TREE command.
- When you insert REAL or DOUBLE values, truncation or padding occurs as needed. No messages appear.
- E-notation is acceptable for all numeric input in SCF operations. The ±n.nnnE±ee format can be used when entering values for INTEGER, DECIMAL, or MONEY items (as well as REAL and DOUBLE) and when specifying constants in functions, where-clauses, and if-clauses. When the software converts a floating point number (E-notation) to fixed format, the result must conform to the picture specified for the INTEGER, DECIMAL, or MONEY item. The E-notation value might not be stored or displayed exactly as entered. For example, the value 10.999877E00 for an INTEGER item becomes 11.

  Input for REAL and DOUBLE items is usually in E-notation, but whole numbers and decimal numbers are acceptable. The output for REAL and DOUBLE values is always standard scientific notation in the form of ±n.nE±ee.
- UNDEFINED input values can contain all 256 EBCDIC characters from X'00' through X'FF'. The values are stored left-justified with trailing binary zero fill if the value is smaller than the picture. Binary zeros are allowed within a value, but an UNDEFINED value containing all binary zeros is considered a missing value (null).

  You can use display format or HEX notation to input UNDEFINED item values (for example, X'E2E8E2E3C5D4'). For more information about entering UNDEFINED values see **Coding UNDEFINED values** on page 8-17.

Examples Specifying BEFORE/AFTER

Examples 1 through 8 show the effect of INSERT TREE commands with before-clauses and after-clauses. In the commands, the newly specified subtree consists of only one record. In general, the specified subtree can include both the top record and its descendant records.

In the first six examples, two data trees are shown. The first tree shows the data tree before the command is executed. Arrows point to the selected records. The second tree shows the data tree with the new records added to it. These records appear in boldface type.

Examples 1 and 2 show an INSERT TREE command where the qualified record is at the same level as the specified record. In Examples 3 and 4, the qualified records are one level above

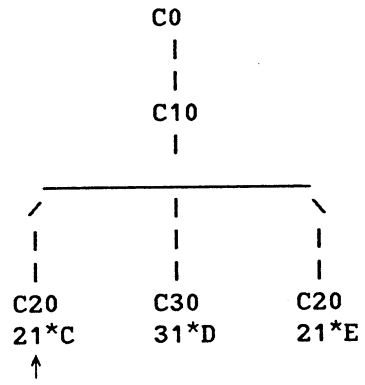
the specified record. In Examples 5 and 6, the qualified records are two levels above the specified record.

In all cases, records similar to the specified record are selected. Notice the effect that the difference in qualification and selection levels has on the number of new records added to the tree. The larger the difference, the higher the chance that one or more of the selected records do not meet the criteria that you have in mind.

All six examples show the position of the new records in relation to their siblings. This position may be important in some applications since it determines the logical order of the records within the tree.

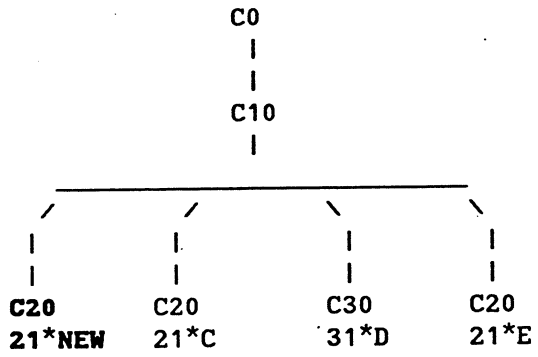
Example 1

This example uses INSERT BEFORE with the qualified record at the same level as the new record.



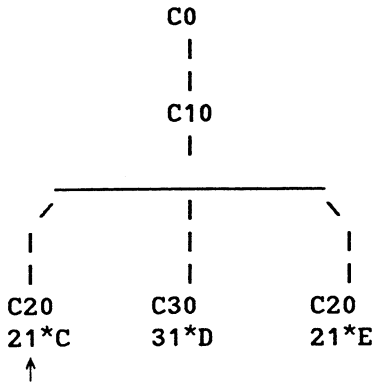

```
INSERT TREE C20 EQ 21*NEW*END* BEFORE C21 EQ C:

- 1 SELECTED RECORD(S) -
```



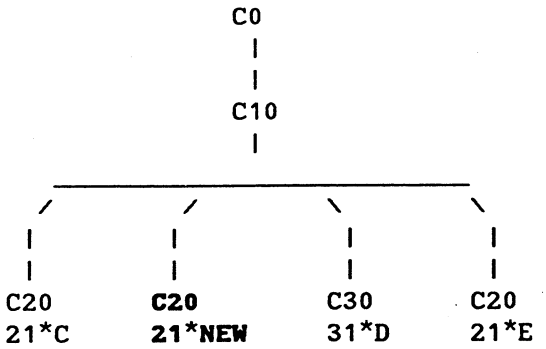
Example 2

This example uses INSERT AFTER with the qualified record at the same level as the new record.



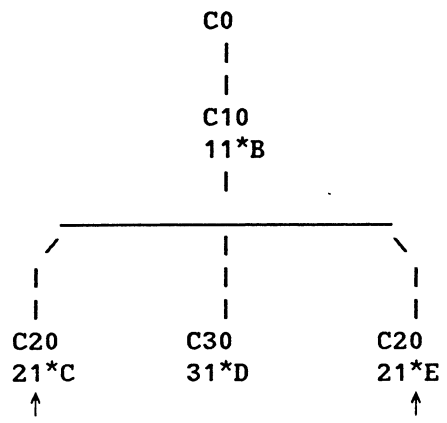

```
INSERT TREE C20 EQ 21*NEW*END* AFTER C21 EQ C:

- 342- 1 SELECTED RECORD(S)
```



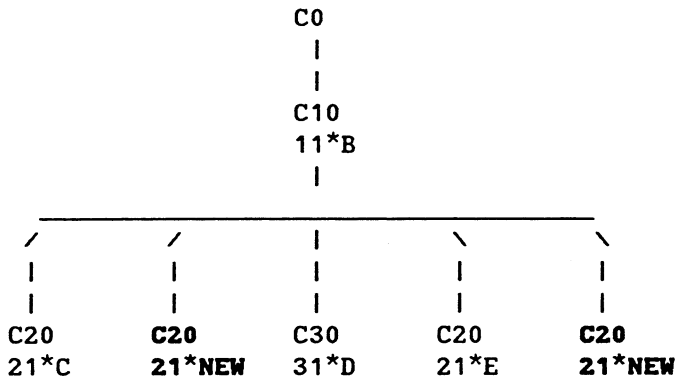
Example 3

This example uses INSERT AFTER with the qualified record one level above the new record.



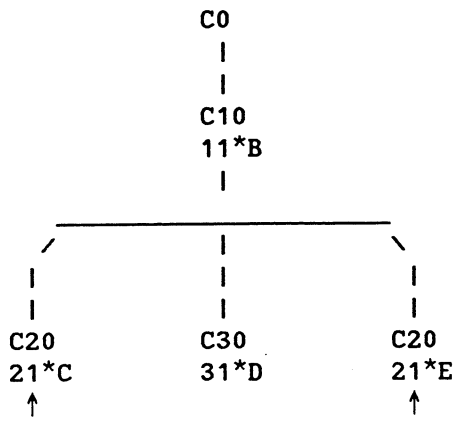

```
INSERT TREE C20 EQ 21*NEW*END* AFTER C11 EQ B:

- 342- 2 SELECTED RECORD(S) -
```



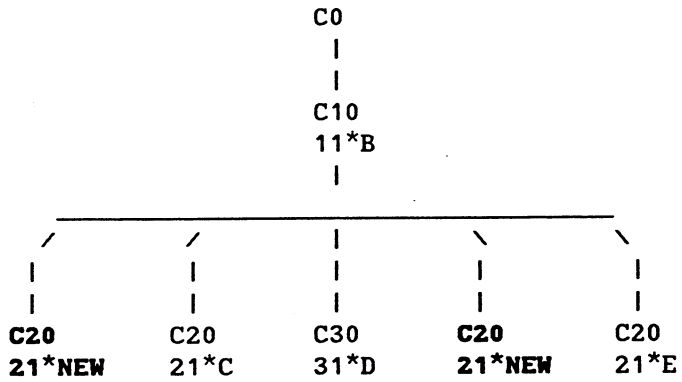
Example 4

This example uses INSERT BEFORE with the qualified record one level above the new record.



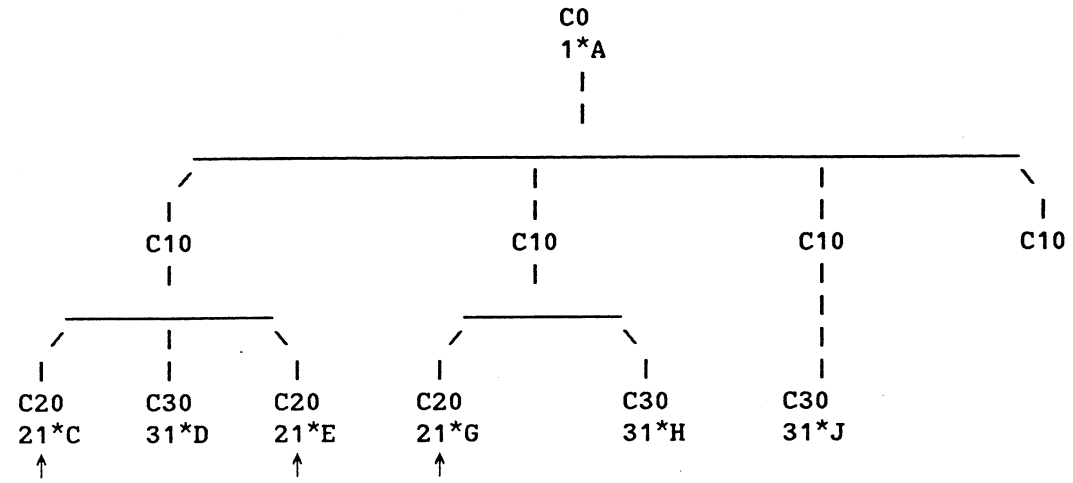

```
INSERT TREE C20 EQ 21*NEW*END* BEFORE C11 EQ B:

- 342- 2 SELECTED RECORD(S)
```



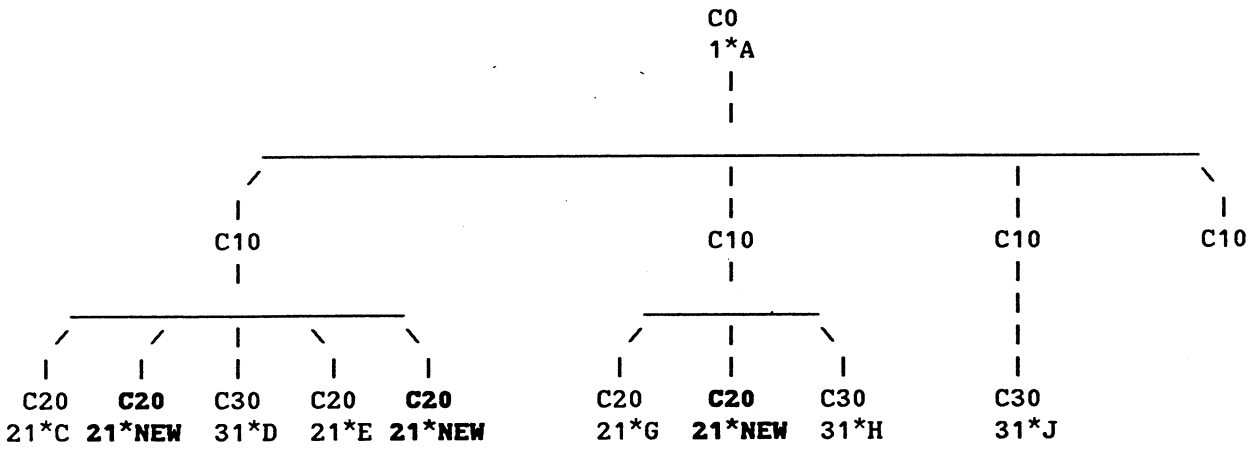
Example 5

This example uses INSERT AFTER with the qualified record two levels above the new record. Notice that no C20 records are inserted under the last two C10 records because they have no descendant C20 records that could be selected.



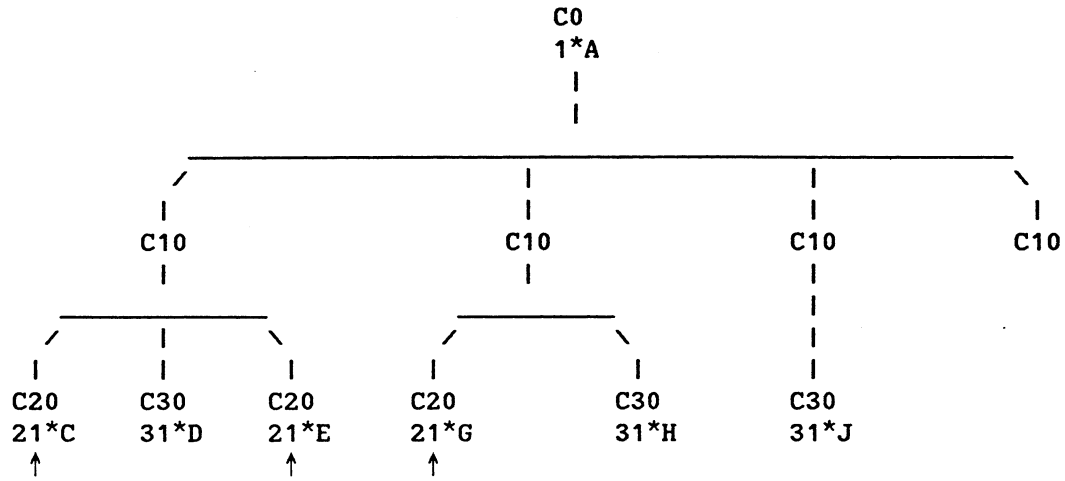

```
INSERT TREE C20 EQ 21*NEW*END* AFTER C1 EQ A:

- 342- 3 SELECTED RECORDS -
```



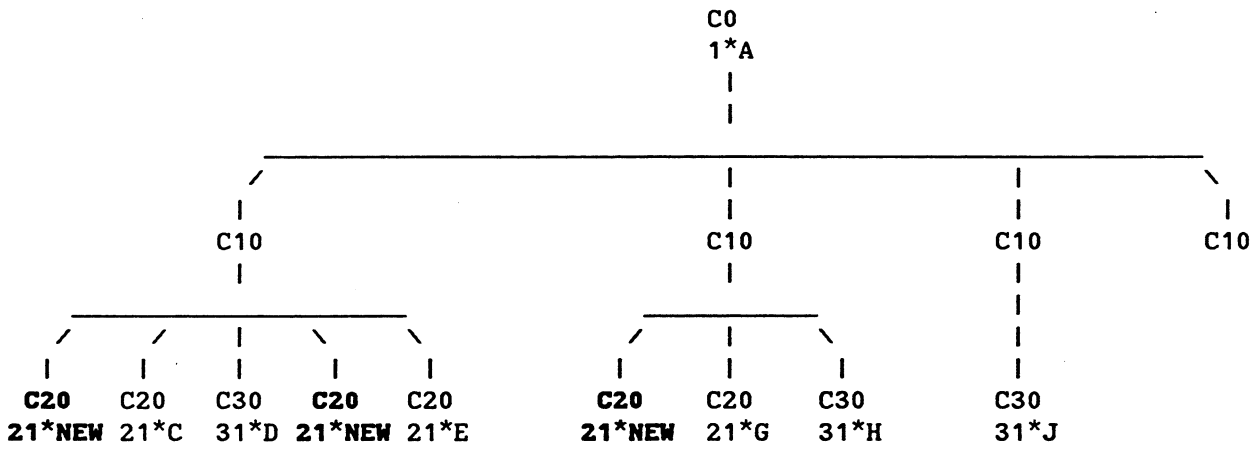
Example 6

This example uses INSERT BEFORE with the qualified record two levels above the new record. Notice that no C20 records are inserted under the last two C10 records because they have no descendant C20 records to be selected.




```
INSERT TREE C20 EQ 21*NEW*END* BEFORE C1 EQ A:

- 342- 3 SELECTED RECORDS-
```



Example 7

If you issue the following command, the software issues an error message. This is because the qualified record (the C30 record with C31 equals D) cannot be used as a reference point for selecting a dissimilar C20 record. The same result is obtained if AFTER is used instead of BEFORE.

```
INSERT TREE C20 EQ 21*NEW*END* BEFORE C31 EQ D:
```

Example 8

The following command inserts a new entry before each logical entry where C1 equals A.

```
INSERT TREE C0 EQ 1*Z*10*11*X*20*21*W*20*21*V*30*31*T*END*
   BEFORE C1 EQ A:
```

Examples Specifying Partial Trace with a Where-Clause

Examples 9 through 17 show the effect of INSERT TREE commands with partial trace and where-clauses. In all commands, the new tree consists of only one record. In general, though, the new tree can include both the top record and its descendant records.

In each example, two data trees are shown. The first tree shows the data tree before the command is executed. Dashed arrows point to the records selected initially. Solid arrows point to the records selected as reference points for insertion. These can be either similar siblings or parents of the new record.

Notice that the number in the SELECTED RECORD(S) message reflects the number of records selected as points of insertion. The second tree shows the data tree with the new records added to it by the command. These records appear in boldface type.

All examples show the position of the new records in relation to their siblings. This position is important as it determines the logical order of the records within the tree.

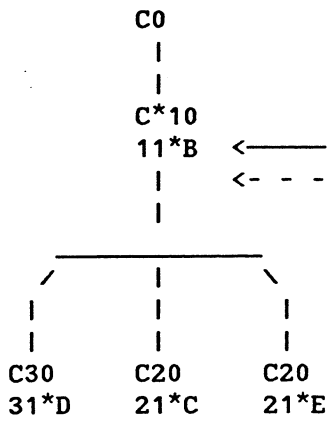
Example 9

This example shows four commands with different where-clauses that have the same effect upon a logical entry. All four commands insert a C20 record first among its siblings (trace: *1). Each command qualifies a different record, but they all select the C10 record to be the parent of the new record, since that C10 record is in the path of the qualified records.

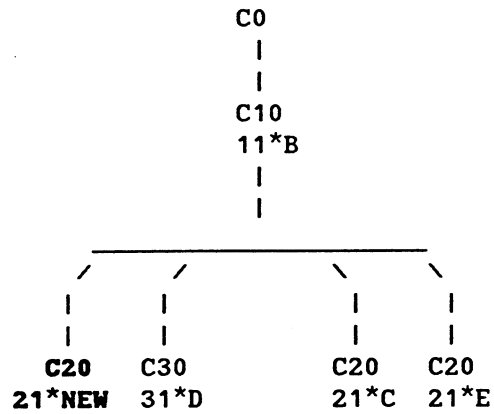
Compare the third command with the command in Example 7. The latter command tells the software to insert the new C20 record before a similar sibling. The software refuses to process the command since the qualified record (C30) is not in the same path as the record specified (C20).

The third command tells the software to insert the new C20 record in the first position among its similar siblings under a C10 record. Since the C10 record is a candidate for selection when a C30 record is qualified by the where-clause, the software selects that C10 record to be the parent of the new C20 record and inserts a new first record under the parent.

Example 9 (continued)

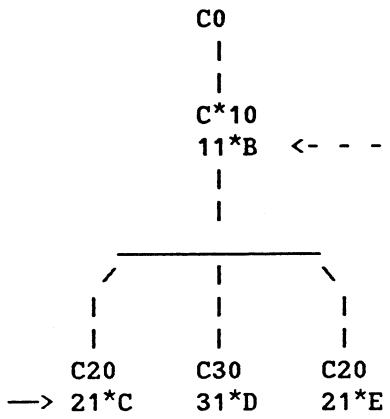



```
INSERT TREE C20 *1 EQ 21*NEW*END* WHERE C21 EQ C:
INSERT TREE C20 *1 EQ 21*NEW*END* WHERE C21 EQ E:
INSERT TREE C20 *1 EQ 21*NEW*END* WHERE C31 EQ D:
INSERT TREE C20 *1 EQ 21*NEW*END* WHERE C11 EQ B:
```

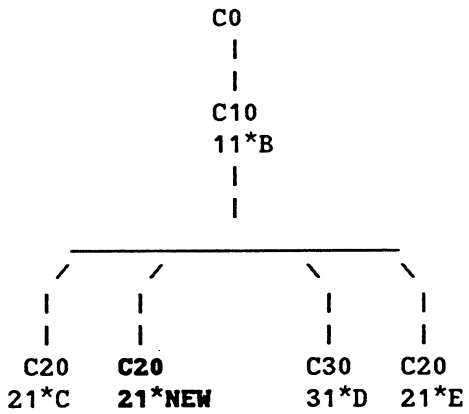


Example 10

This example also shows four commands with different where-clauses. Since the specified record is at level 2 and there is one number in the trace notation, record C10 at level 1 is selected initially. In this case, all four commands insert the new C20 record in second place among its similar siblings (trace *2).

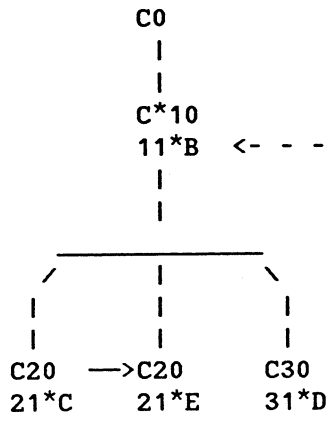



```
INSERT TREE C20 *2 EQ 21*NEW*END* WHERE C21 EQ C:
INSERT TREE C20 *2 EQ 21*NEW*END* WHERE C21 EQ E:
INSERT TREE C20 *2 EQ 21*NEW*END* WHERE C31 EQ D:
INSERT TREE C20 *2 EQ 21*NEW*END* WHERE C11 EQ B:
```

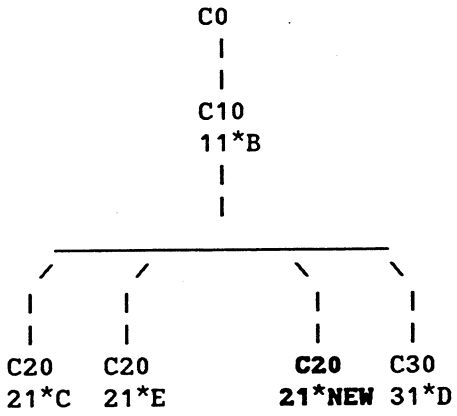


Example 11

The four commands in this example insert the new C20 record in the last place (trace *0) among its similar siblings, that is, after the last existing similar sibling, not as the last record under the parent.

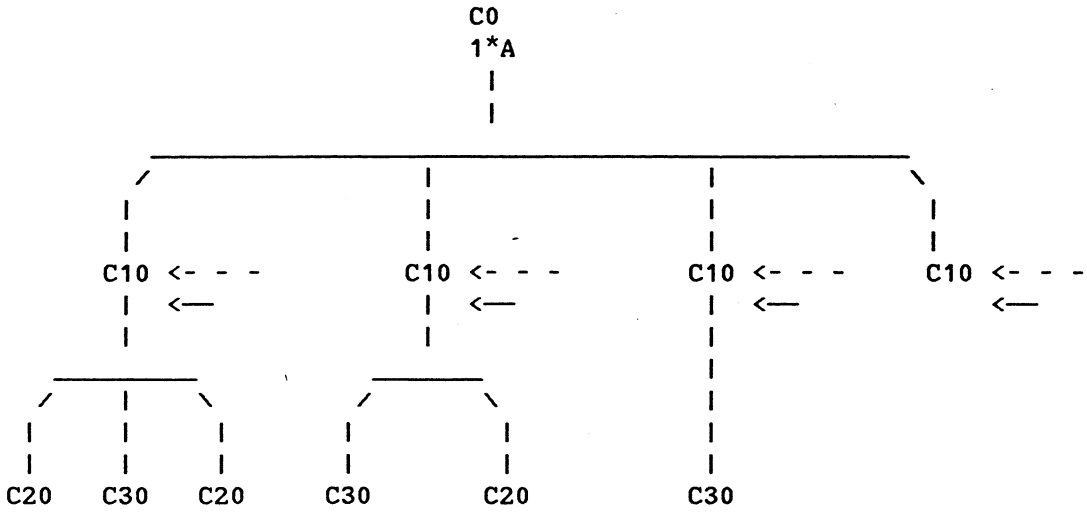



```
INSERT TREE C20 *0 EQ 21*NEW*END* WHERE C21 EQ C:
INSERT TREE C20 *0 EQ 21*NEW*END* WHERE C21 EQ E:
INSERT TREE C20 *0 EQ 21*NEW*END* WHERE C31 EQ D:
INSERT TREE C20 *0 EQ 21*NEW*END* WHERE C11 EQ B:
```



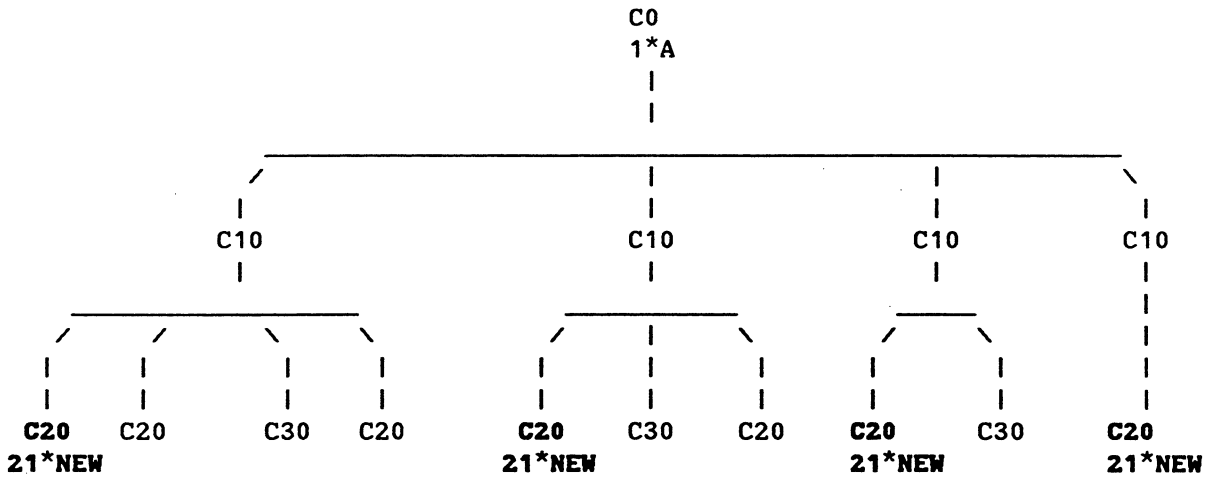
Example 12

In this example, the where-clause qualifies the C0 record; all four C10 records are selected to be parents of the new C20 record. The C20 records are inserted in the first position under each selected C10 record.



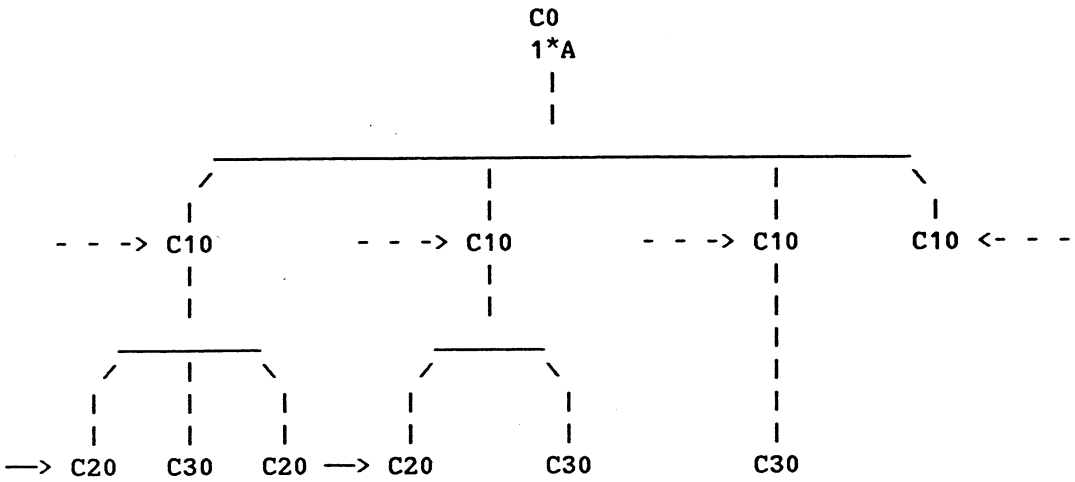

```
INSERT TREE C20 *1 EQ 21*NEW*END* WHERE C1 EQ A:

- 342- 4 SELECTED RECORD(S) -
```



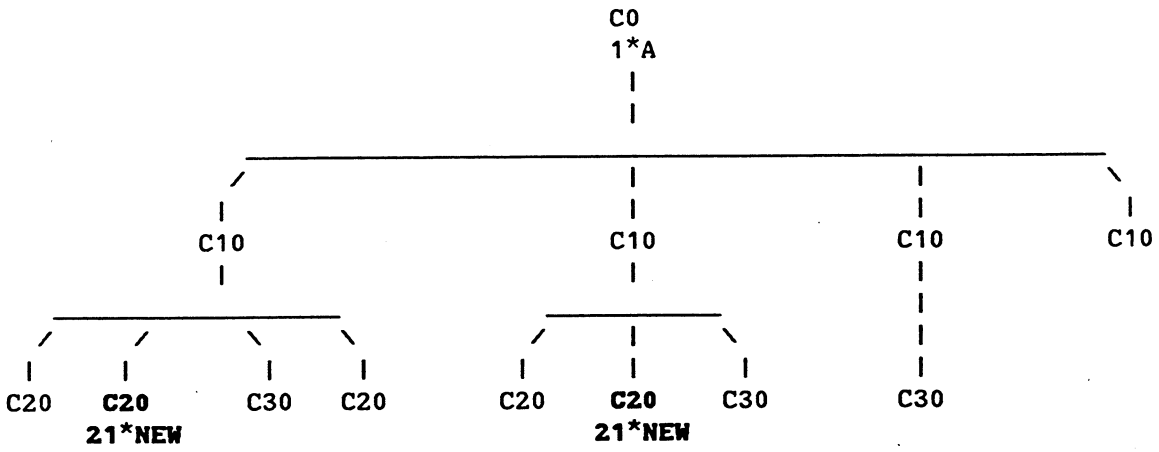
Example 13

In this example, the where-clause also qualifies the C0 records, and all four C10 records are selected. Since the command specifies that the new C20 record is to be placed in the second position among its similar siblings, only the existing C20 records are selected as points of insertion.



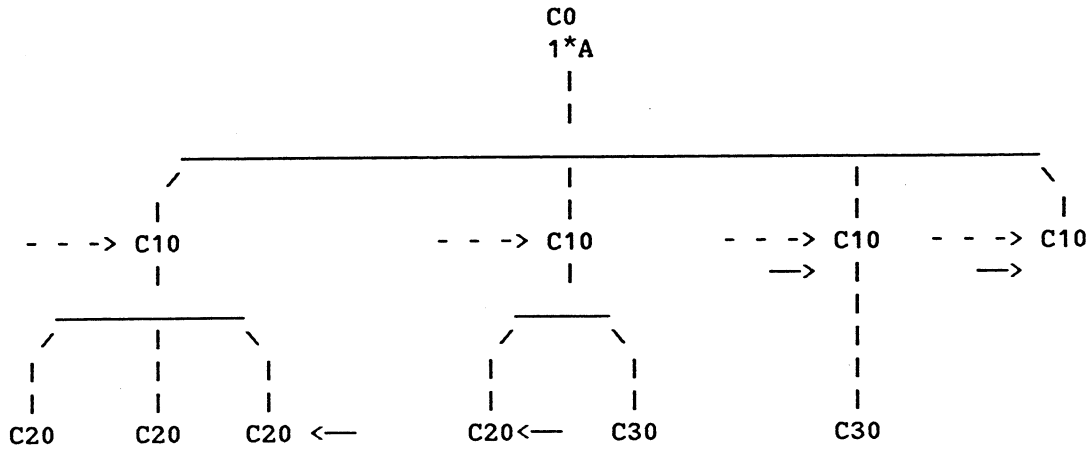

```
INSERT TREE C20 *2 EQ 21*NEW*END* WHERE C1 EQ A:

- 342- 2 SELECTED RECORD(S) -
```



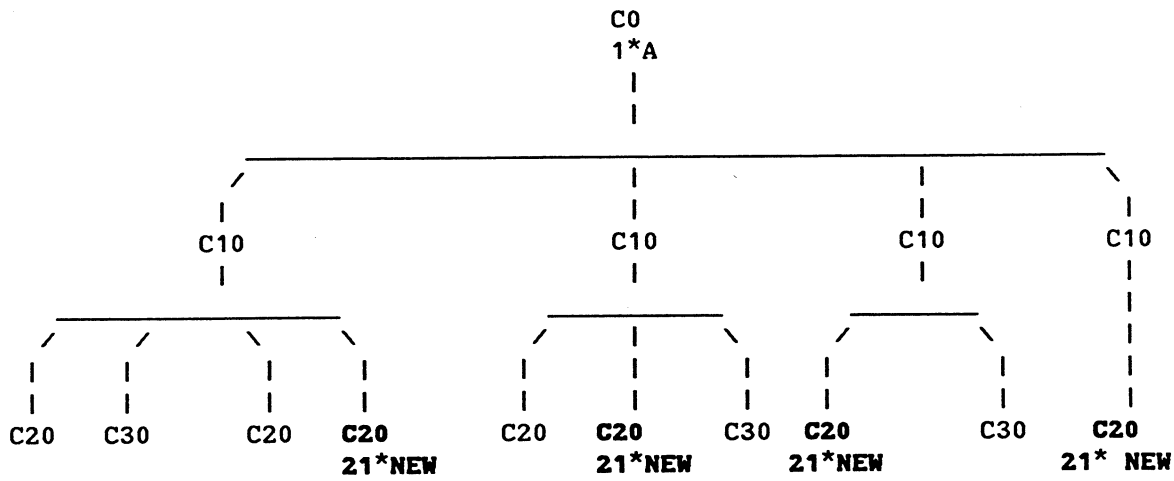
Example 14

In this example, the where-clause qualifies the C0 record; all four C10 records are selected to be parents of the new C20 record. The C20 records are inserted under each selected C10 record. If there is a similar sibling, the C20 record is inserted after it. Otherwise, it is inserted as the first record under the selected C10 record.



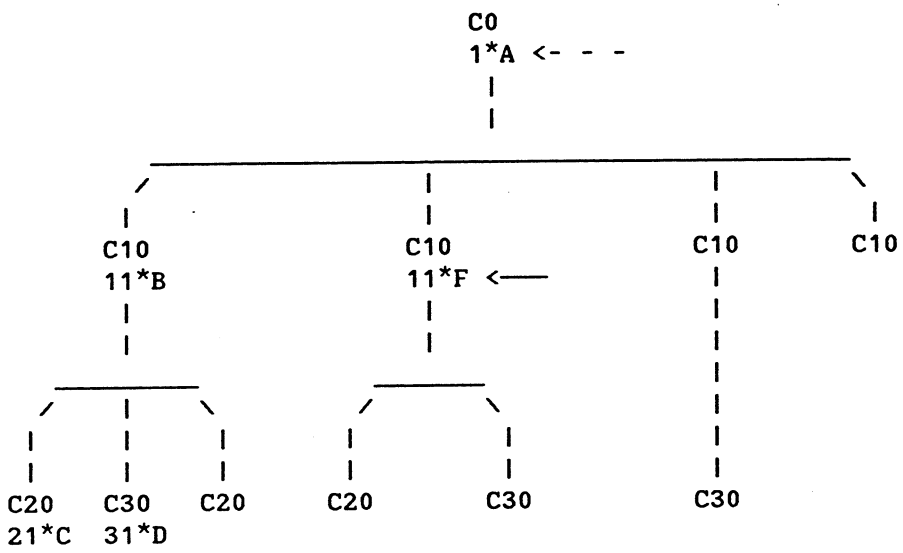

```
INSERT TREE C20 *0 EQ 21*NEW*END* WHERE C1 EQ A:

- 342- 4 SELECTED RECORD(S) -
```

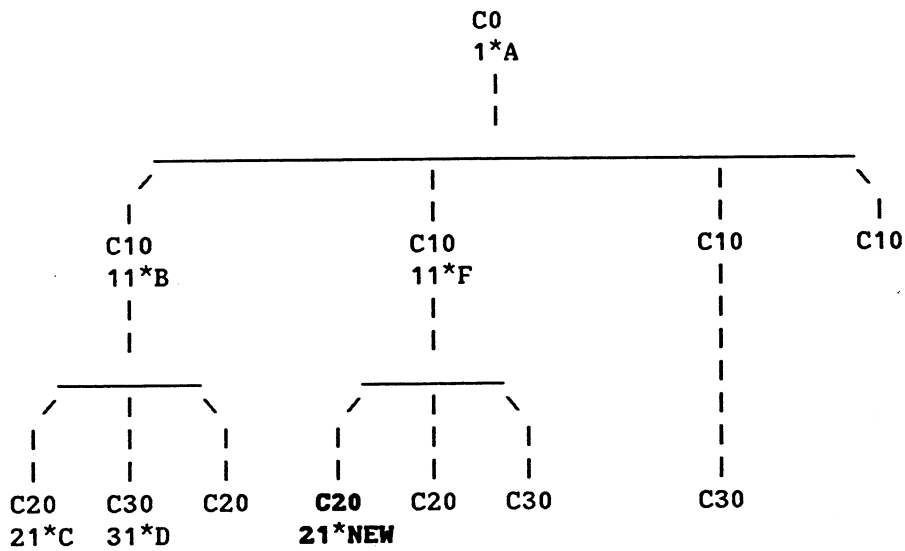


Example 15

In this example, the commands specify that the C20 record be placed in first position under C10 records, which in turn are to be in second place under selected C0 records. Since the specified record is at level 2 and there are two numbers in the trace notation, level 0 records are selected initially, allowing qualification to be at any level in any path.

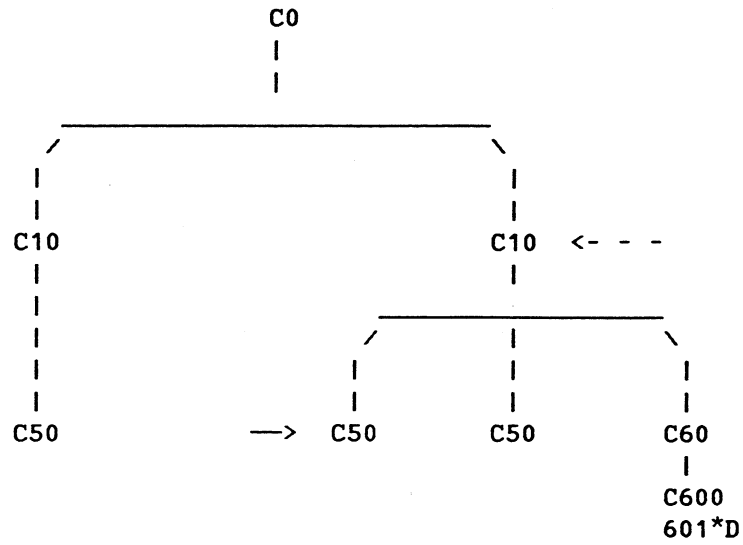



```
INSERT TREE C20 *1*2 EQ 21*NEW*END* WHERE C11 EQ B:
INSERT TREE C20 *1*2 EQ 21*NEW*END* WHERE C11 EQ F:
INSERT TREE C20 *1*2 EQ 21*NEW*END* WHERE C21 EQ C:
INSERT TREE C20 *1*2 EQ 21*NEW*END* WHERE C21 EQ D:
```

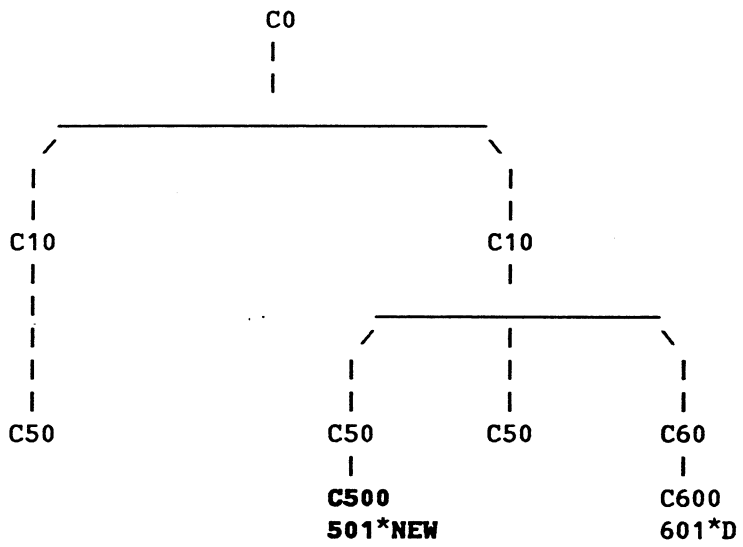


Example 16

In this example, the command specifies that the C500 record be placed in first position under C50 records, which in turn are to be in first position under selected C10 records. Thus, the C10 record, which is an ancestor of the qualified C600 record, is selected, and the new C500 record is placed under its first C50 child.




```
INSERT TREE C500 *1*1 EQ 501*NEW*END* WHERE C601 EQ D:
```



Example 17

This command produces the message given below, because the command specifies that the C500 record be placed in first position under selected C50 records. Since C50 records are in a different schema path than C600 records, the software cannot honor the request.

```
INSERT TREE C500 *1 EQ 501*NEW*END* WHERE C601 EQ D:

- 342- ACTION-CLAUSE COMPONENTS MUST BE IN WHERE-CLAUSE PATH -
```

Examples Specifying Full Trace and No Where-Clause

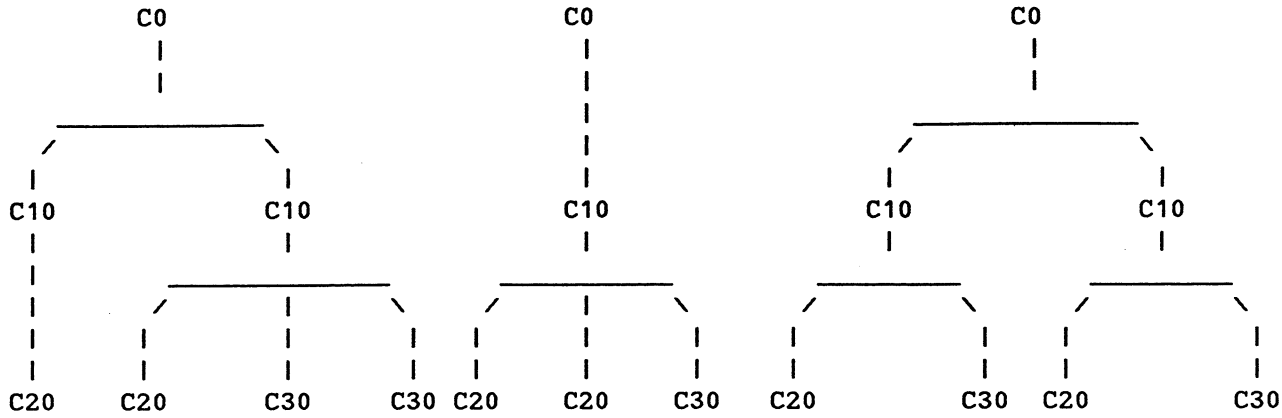
Examples 18 and 19 show the effect of an INSERT TREE command with a full trace.

In each example, the database is shown before and after the command is executed. The records added to the database by the command appear in boldface type.

Both examples show the position of the new records in the database. This position may be important in some applications since it determines the logical order of the records in the database.

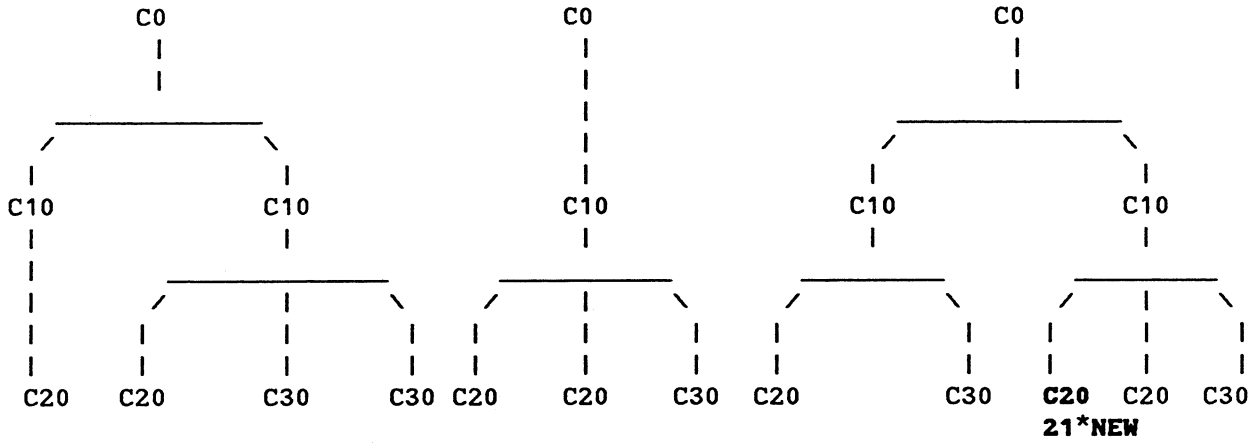
Example 18

This example shows an INSERT TREE command that inserts a C20 record in first position (trace *1) under the second (trace *2) C10 record of the third (trace *3) logical entry.



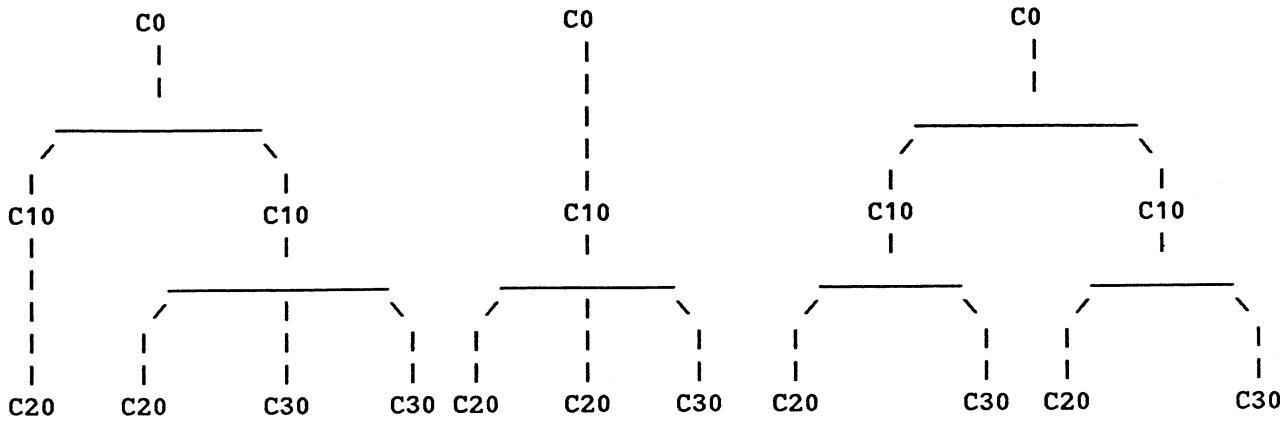

```
INSERT TREE C20 *1*2*3 EQ 21*NEW*END*:

- 342- 1 SELECTED RECORD(S) -
```



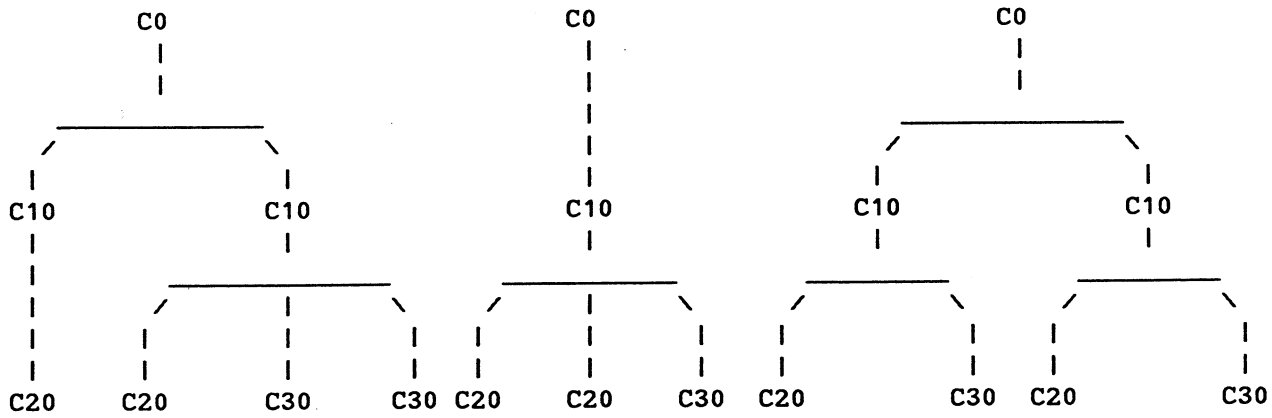
Example 19

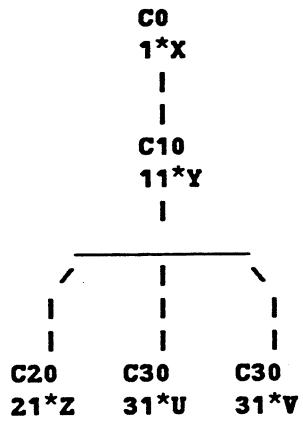
This example shows an INSERT TREE command that inserts a new entry after the last (trace *0) logical entry in the database. This database consists of only three entries.




```
INSERT TREE C0 *0 EQ 1*X*10*11*Y*20*21*Z*30*31*U*30*31*V*END*:

- 342- 1 SELECTED RECORD(S) -
```





MOVING DATA TREES: MOVE TREE

The MOVE TREE command moves a data record and its descendants from one position in a database to another position in the same database. It is not a copy operation.

The data tree to be moved can be an entire logical entry or any subtree in an entry. The MOVE TREE command

- moves entire trees or subtrees from one position to another.
- modifies appropriate pointers to accomplish the move without changing the physical position of the data records in the database or the index.
- is the least expensive and most efficient way of moving data trees in a database. Only the Hierarchical Table pointers are changed. Physical movement of data and updating of the index are completely bypassed.

The MOVE TREE command allows you to move a data tree of records from one logical position to another in a single command without changing the physical structure of the database tables or index. The MOVE TREE command is a fast, single operation that involves changing only the appropriate pointers in the Hierarchical Table.

Other methods (for example, REMOVE TREE and INSERT TREE or an UNLOAD followed by an APPEND TREE with *DATA*) require a two- or three-step procedure that involves time and processing to do the actual movement of the data records and to update the indexes of any key items.

The MOVE TREE capability is a one-to-one move. That is, a single data tree is logically moved from one position to a another position. The data tree cannot be copied to multiple new positions.

The MOVE TREE command has two where-clauses (or partial/full traces). The first where-clause (or trace) selects the top node (record) of the tree or subtree to be moved. The second where-clause (or trace) selects one point of attachment for the tree to be moved. That is, the first where-clause establishes a "from" position, and the second where-clause establishes a "to" position.

The selection criteria must qualify only one record for the "from" position and only one record for the "to" position, or an error message will be issued. If the "from" position is not exactly one selected record, the "to" where-clause is not executed and a set of informative error messages is issued.

Executing the MOVE TREE command causes a logical change in the data record pointers, but no physical changes to the data structure or index. The software carries out this change by switching the pointers in the Hierarchical Table entries to reflect the removal and insertion of the top node of the data tree affected.

Existing INSERT TREE command rules govern the attachment of the data tree at its new position.

Formats

```
MOVE TREE record |where-clause                    TO  |before-clause  :
                 |partial-trace where-clause          |after-clause
                 |full-trace                           |partial-trace where-clause
                                                       |full-trace
```

record is a schema record name or component number; the target record.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

before-clause is the same as a where-clause except that the keyword BEFORE replaces the keyword WHERE. See Chapter 24, "The Where-Clause," for details.

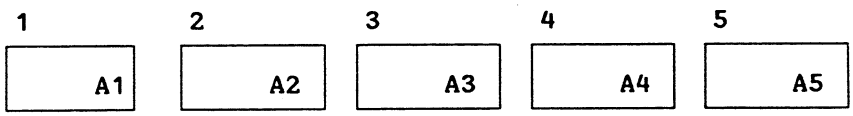
after-clause is the same as a where-clause except that the keyword AFTER replaces the keyword WHERE. See Chapter 24, "The Where-Clause," for details.

Abbreviations:

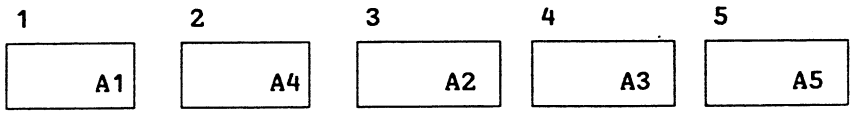
AFTER = AF
BEFORE = BE
MOVE TREE = MT

Considerations

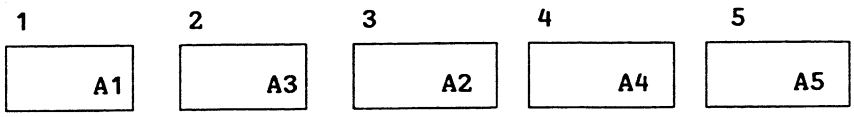
- You cannot issue the MOVE TREE command for a damaged database.
- You must have U-authority for the records specified in the action-clause of a MOVE TREE command.
- The use of SAME is acceptable for either or both of the where-clauses in a MOVE TREE command. The DYNAMIC and STATIC settings are honored in the same way as for any other consecutive where-clauses. After a MOVE TREE command, the SAME file contains the results of the specified "to" where-clause if SAME is set to dynamic.
- The MOVE TREE command uses the same rules as the INSERT TREE command for establishing the parent of the sibling at the place of attachment, except that the "to" position is limited to only one selected record.
- If Security by Entry is in effect for the database, a secondary password holder can move a subtree from one logical entry to another, provided you know both entry key values. The last where-clause used in setting up the "to" position determines the current entry key after the MOVE TREE command completes processing.
- The keyword TO must precede the second where-clause (or trace).
- Each where-clause can be forced to select only one record (unless no records qualify) by setting LIMIT 1, TRUNCATE. In effect, this guarantees that the MOVE TREE command will execute. However, the specific record selected for the "from" and/or the "to" position will be the first one in physical order. Therefore, do not use the TRUNCATE option with LIMIT for the MOVE TREE command because you have no control over which tree will be moved or where the point of insertion will be. Use LIMIT 1 and the exact selection criteria to select the correct "from" and "to" records.
- If the "to" position uses trace notation (full or partial), then the software inserts the selected data tree before the "to" position record.
- If the "from" position and the "to" position are both in the same sibling chain and trace notation is used to establish the "to" position, then the final trace position of the "from" record depends upon whether it initially precedes or follows the "to" record. For example, suppose a sibling chain of five level 0 records exists before processing any MOVE TREE command:



The command MOVE TREE ENTRY *4 TO *2 would yield the following results:



However, the command MOVE TREE ENTRY *2 TO *4 (for the original chain of siblings) would yield these results:

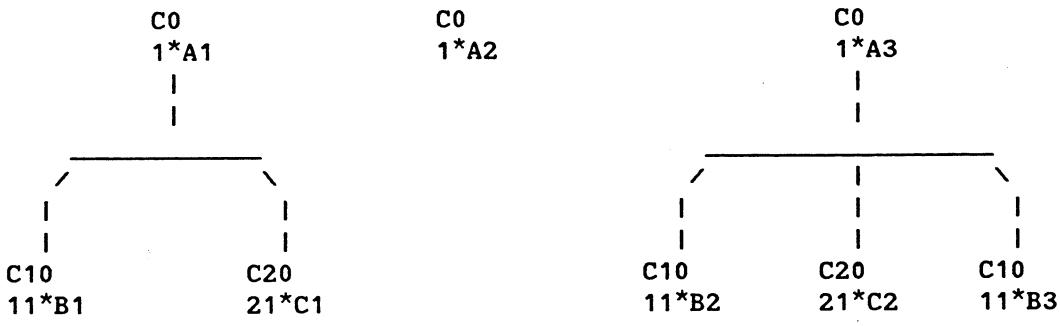


- If each where-clause selects exactly one record, the MOVE TREE command is processed, and only the following messages are issued (one for each where-clause):

```
-342-  1 SELECTED RECORD(S)  -
-342-  1 SELECTED RECORD(S)  -
```

Examples

Each of these examples uses the three hypothetical logical entries shown below. The level 0 record has one item (C1). Record C10 and record C20 are at level 1; each has one item, C11 and C21, respectively.



Because you can specify either a full trace or a partial trace for either where-clause, the "from" and "to" positions can be specified many ways. Each example shows several variations that accomplish the same results.

Example 1

To place the entry whose level 0 record ("from" record) contains C1 equal to A1 between A2 and A3, you could use any of the following commands:

```
MOVE TREE ENTRY WHERE C1 EQ A1 TO AFTER C1 EQ A2:
MOVE TREE C0*1 TO BEFORE C11 EQ B2:
MOVE TREE C0 WHERE C21 EQ C1 TO BEFORE C1 EQ A3:
MOVE TREE ENTRY WH C1 EQ A1 TO *3:
MOVE TREE C0*1 TO *3:
```

Example 2

To move the level 1 record B1 under the level 0 record A2, use any of the following equivalent commands:

```
MOVE TREE C10 WHERE C11 EQ B1 TO *1 WHERE C1 EQ A2:
MOVE TREE C10 WHERE C11 EQ B1 TO *0 WHERE C1 EQ A2:
MOVE TREE C10 *1*1 TO *0*2:
MOVE TREE C10 *1*1 TO *1*2:
```

Example 3

This example shows one way to place the C1 record between the B2 and C2 records.

```
MOVE TREE C20 WHERE C21 EQ C1 TO BEFORE C21 EQ C2:
```

Example 4

This example shows one way to place the C1 record before the B2 record.

```
MOVE TREE C20 WH C21 EQ C1 TO *1 WHERE C1 EQ A3:
```

Example 5

This example shows one way to move the first data record under A3 to the end of its level 1 sibling chain under A3 (that is, to make the first level 1 record be the last one).

```
MOVE TREE C10 *1 WHERE C1 EQ A3 TO *0 WHERE SAME:
```

Example 6

Suppose Security by Entry is in effect, and a secondary password holder knows the entry key values for C1 for both the first logical entry (A1) and the third logical entry (A3). To move the B1 record from the A1 entry to the A3 entry, the following command is acceptable:

```
MT C10 WH C1 EQ A1 AND C11 EQ B1 TO *0 WH C1 EQ A3:
```

REMOVING DATA TREES: REMOVE TREE

The REMOVE TREE command removes data trees from existing logical entries; that is, it removes selected records and their descendants. The REMOVE TREE command

- removes data trees
- can include a partial trace in conjunction with a where-clause to pinpoint the location of selected records
- can include a full trace to select data trees for removal
- cannot remove values only (see the REMOVE command to remove values).

The REMOVE TREE command always specifies a target record. The where-clause (if present) and trace notation (if present) select records that are occurrences of the target record.

When the command is executed, these selected records and their descendants are removed from the database. By specifying selection of ancestor records, entire subtrees can be removed without regard to their contents.

After each tree is removed, the remaining trees are relinked to close the gap at the level of the specified record. For example, when C10 is the parent record of three C100 records and the second C100 record is removed by a REMOVE TREE command, the third C100 record becomes the second. The space in the Data Table occupied by the removed records becomes reusable space, although it is not immediately used if more than one user is accessing the database in a Multi-User environment.

Formats

```
REMOVE TREE  record  full-trace  :
```

```
REMOVE TREE  record  [partial-trace]  where-clause  :
```

record is a schema record name or component number; the target record.

full-trace is a list of one or more positive integers or 0, each preceded by the system separator (for example, *1*4*0). In a left-to-right scan, each integer designates the position of the child record under its parent at each level. The rightmost integer designates the level 0 position. Zero means last.

partial-trace is trace notation in which the rightmost integer designates a position other than level 0.

where-clause selects specific data records. See Chapter 24, "The Where-Clause," for details.

Considerations

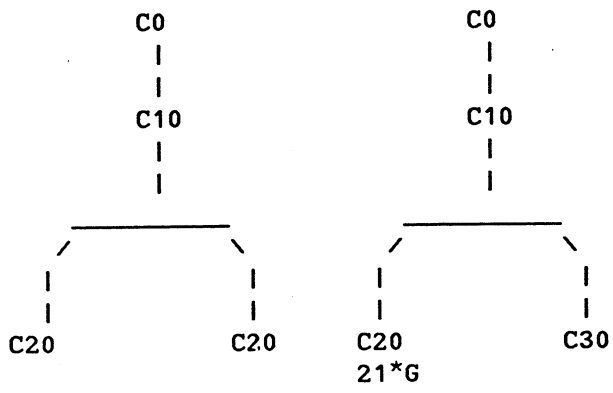
- You cannot issue the REMOVE TREE command for a damaged database.
- You must have U-authority for records specified in the action-clause of a REMOVE TREE command.
- You cannot use Collect File items in the action-clause of a REMOVE TREE command.

Examples

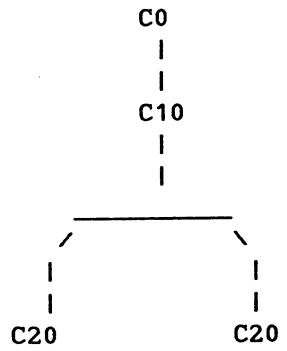
In each of the following examples, the figure above the command(s) shows how the data tree(s) appears before the command is executed. The second figure represents the data tree(s) after the command is executed. Any data records that were removed from the database are deleted from the figure.

Example 1

In this example, the database consists of only two logical entries. Both commands remove the second (or last) entry.

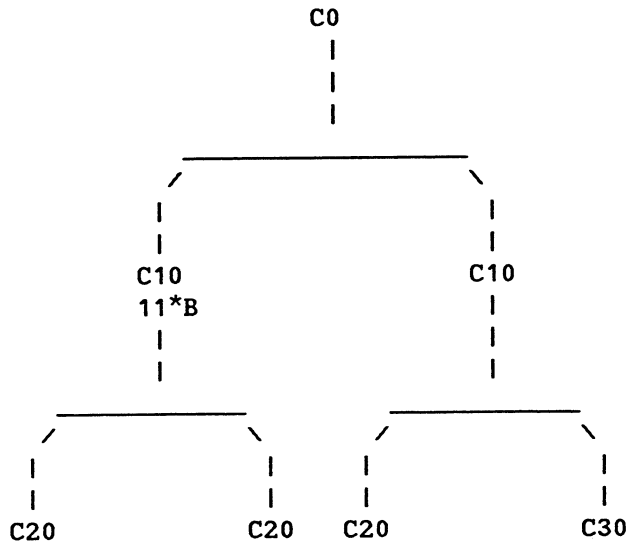



```
REMOVE TREE ENTRY WHERE C21 EQ G:
REMOVE TREE ENTRY *0:
```

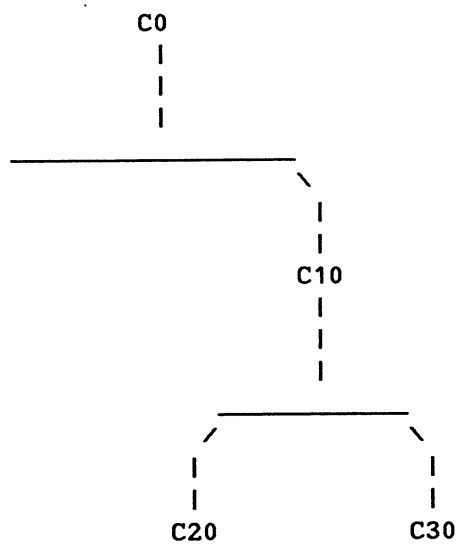


Example 2

This example shows a REMOVE TREE command that removes one C10 record and its descendants from the database.

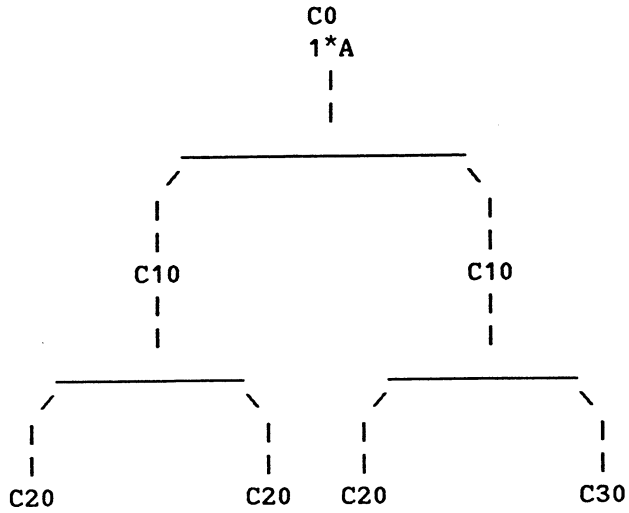



```
REMOVE TREE C10 WHERE C11 EQ B:
```

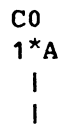


Example 3

This example shows a REMOVE TREE command that removes all C10 records and their descendants from the database.

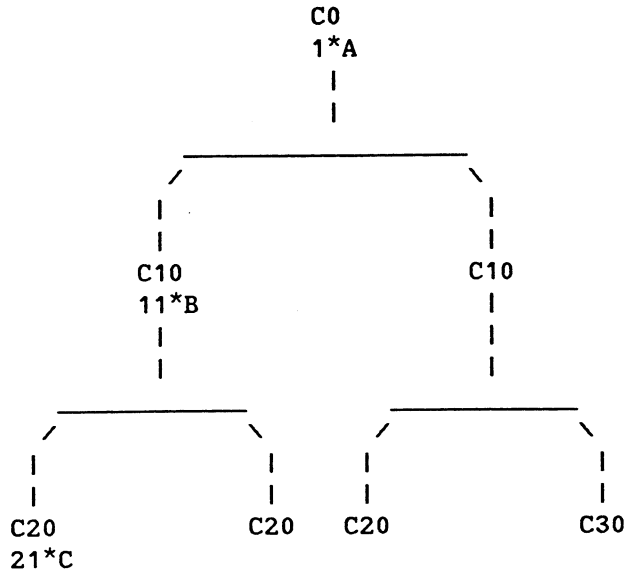



```
REMOVE TREE C10 WHERE C1 EQ A:
```

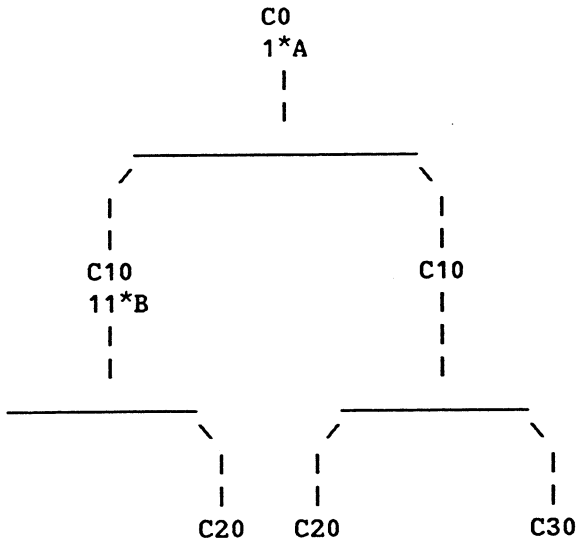


Example 4

This example shows two REMOVE TREE commands. Both have the same effect on the logical entry; they remove one C20 record. The first command includes only a where-clause qualifying the C20 record to be removed; the second command includes a where-clause qualifying a C10 record and a partial trace selecting the first C20 record under the C10 record qualified by the where-clause.

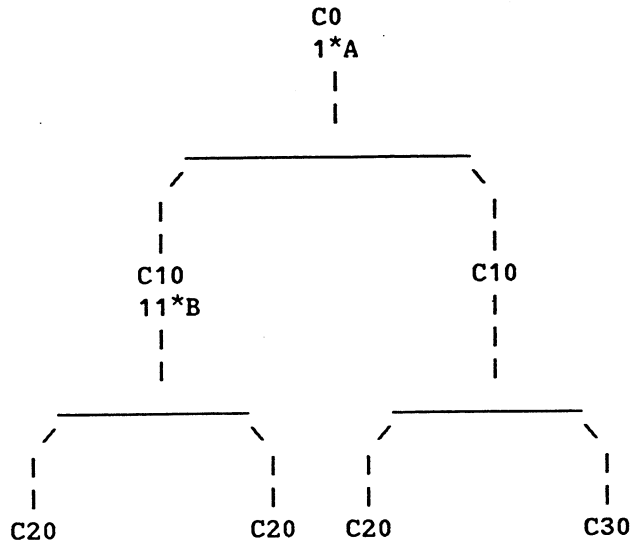



```
REMOVE TREE C20 WHERE C21 EQ C:
REMOVE TREE C20 *1 WHERE C11 EQ B:
```

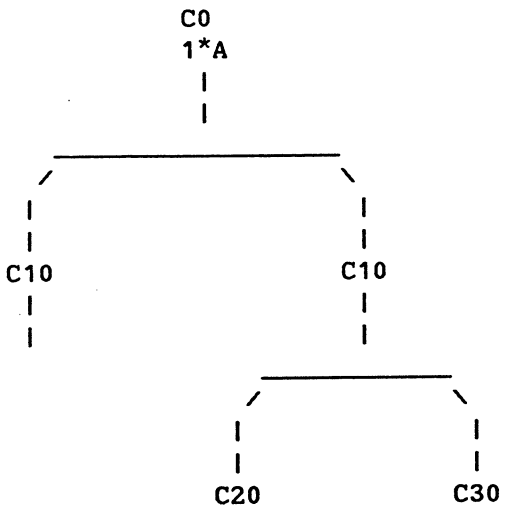


Example 5

This example shows a REMOVE TREE command that removes all C20 records under the C10 record qualified by the where-clause.

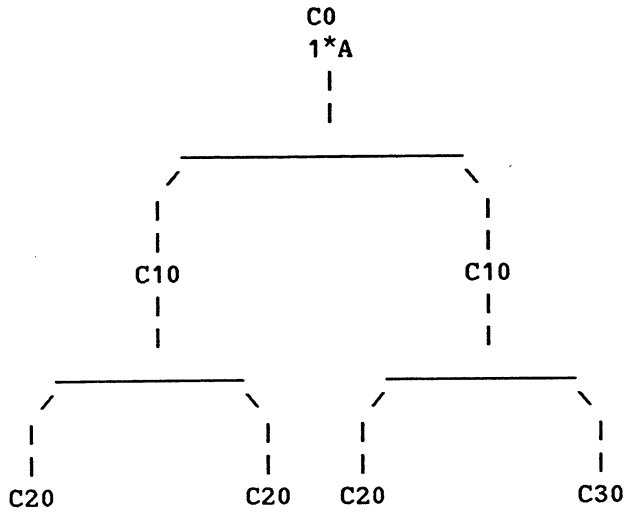



```
REMOVE TREE C20 WHERE C11 EQ B:
```

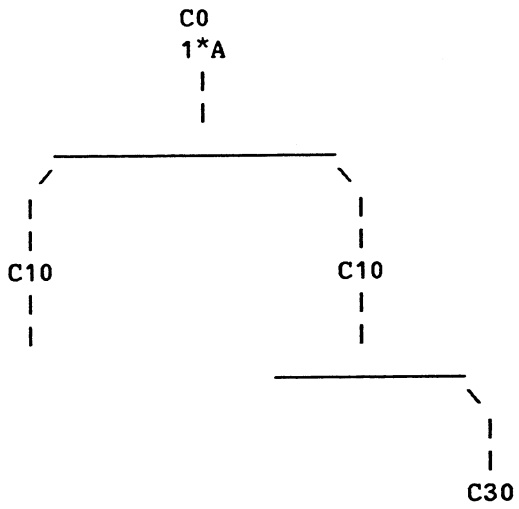


Example 6

This example shows a REMOVE TREE command that removes all C20 records under the C0 record qualified by the where-clause.

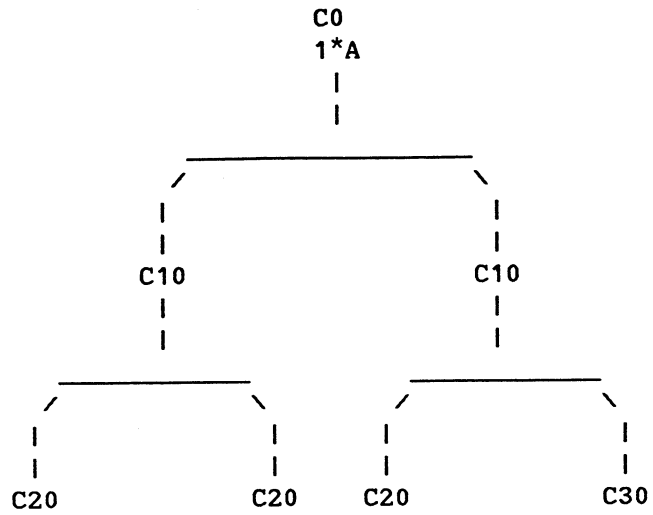



```
REMOVE TREE C20 WHERE C1 EQ A:
```

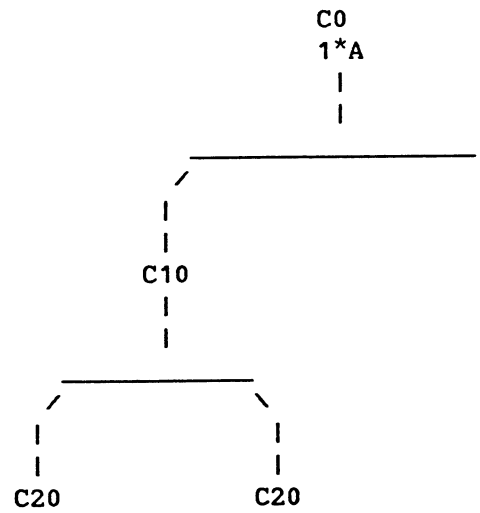


Example 7

This example shows a REMOVE TREE command that removes the C10 record (and its descendants) that is in second position under the C0 record qualified by the where-clause.

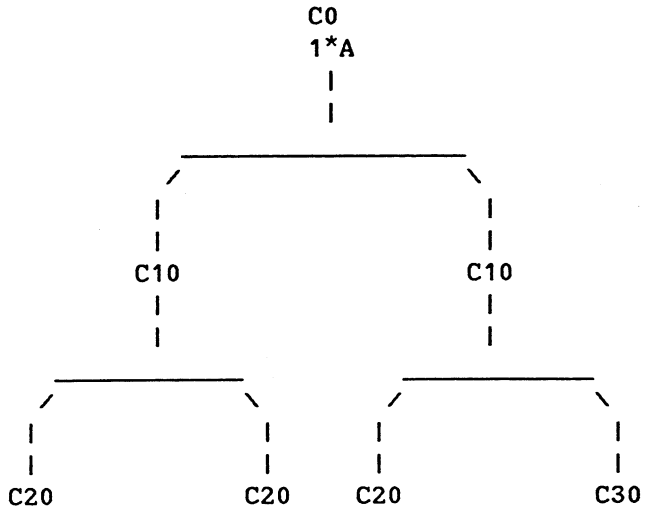



```
REMOVE TREE C10 *2 WHERE C1 EQ A:
```

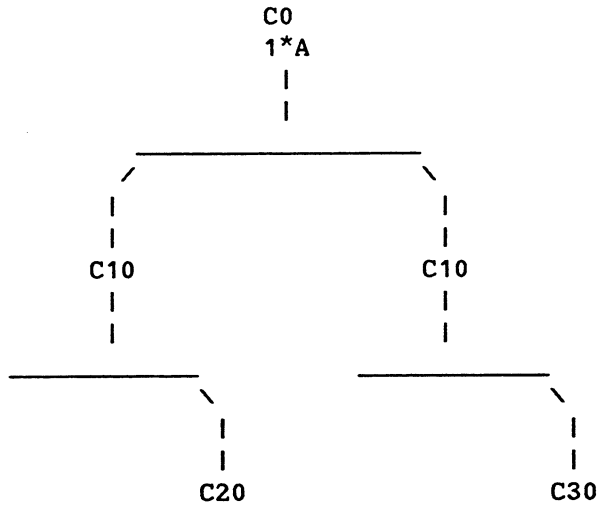


Example 8

This example shows a REMOVE TREE command that removes the first C20 record under each C10 record that is a descendant of the C0 qualified by the where-clause.

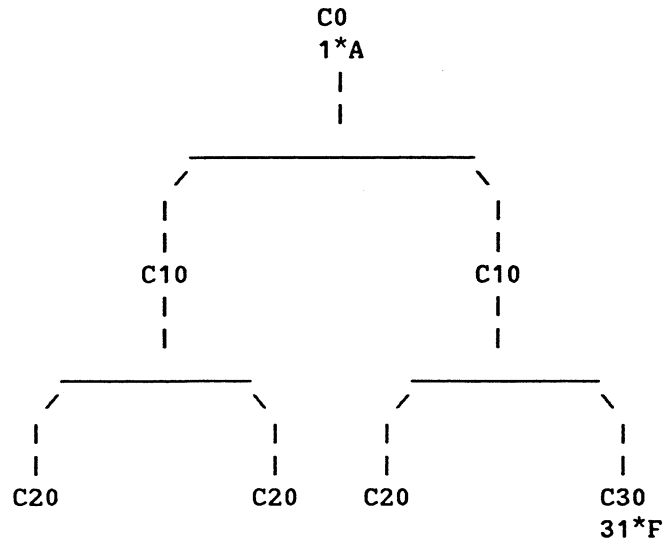



```
REMOVE TREE C20 *1 WHERE C1 EQ A:
```

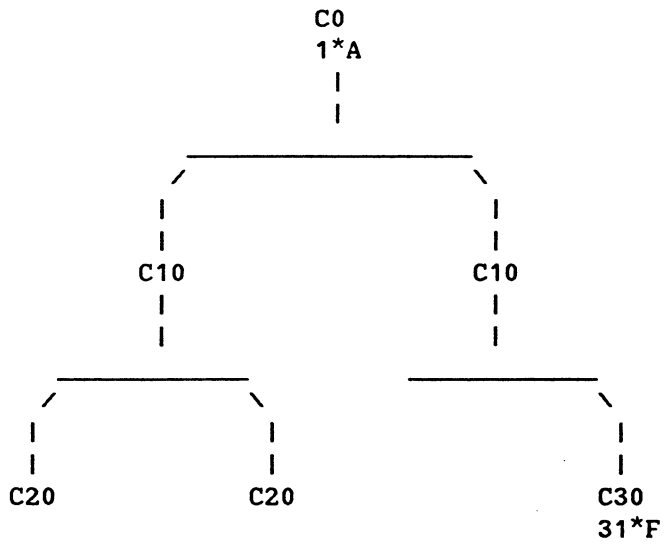


Example 9

This example shows two REMOVE TREE commands. Both include a where-clause and partial trace, and both select the same C20 record for removal.




```
REMOVE TREE C20 *1 WHERE C31 EQ F:
REMOVE TREE C20 *1*2 WHERE C1 EQ A:
```



Example 10

If you issue the command shown below, the software displays an error message. The software cannot process the command because the qualified record (C30) is in a different schema path than the specified record.

```
REMOVE TREE C20 WHERE C31 EQ F:
```

Compare this command with the first command in Example 9. In that case, the trace notation establishes selection for record C10 at level 1. Qualification (via record C30) and normalization to level 1 select one C10 record, and the command removes the C20 record.

Chapter 22

Creating a Logical Unit of Work: FRAME and END FRAME

Format 22-2
Considerations 22-2

When the Rollback Log is enabled, the FRAME and END FRAME commands allow you to establish beginning and ending database synchpoints. Establishing synchpoints around update commands creates a user-specified *logical unit of work*.

Logical units of work are used with Coordinated Recovery. When you issue a FRAME command, you are placing a global hold on the database, and no other user can place a hold on the database or issue an update command.

The database synchpoints created by the FRAME and END FRAME commands permit the software to protect a database against damage by being able to recover the database to the last valid cycle boundary.

FRAME and END FRAME are the only commands that can terminate a logical unit of work where the FRAME command created a beginning database synchpoint.

When a FRAME command has been issued to begin a logical unit of work and a global hold is placed on the database, any other Self-Contained Facility user who tries to issue a FRAME command and begin a logical unit of work is suspended. The user is suspended in the thread unless the suspension would be in the last thread. One thread must remain available in order to avoid a deadlock.

If only the last thread is available, the user is suspended out of the thread. This situation could severely impact a Multi-User environment that is not using multiple threads. Although Rollback does not tie up a thread, there is no way to predict whether a Self-Contained Facility session will request an update that could cause a deadlock on a database with a hold in effect.

For more information about synchpoints, logical units of work, and Coordinated Recovery, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

Whether the Rollback Log is enabled or disabled, the FRAME command causes all user local holds to be dropped. Then a global hold is placed on the database until an END FRAME command is issued. In either case, this allows update commands to be issued without intervention by other users. If the Rollback Log is disabled, the software also issues a warning message.

If the software cannot grant a global hold (for example, if the database is currently in use) and you are a TP user, you receive a message, and the rest of the TP segment is ignored.

When a logical unit of work is terminated, all modified pages for the database are cleared to disk before the damage flag is reset.

Format

```
FRAME [/IMMEDIATE/]  :

END FRAME:
```

FRAME begins a logical unit of work. The FRAME command also ends the previous logical unit of work, if any. Ending a logical unit of work with the FRAME command commits all updates but does not reset the Rollback Log.

END FRAME ends a logical unit of work, resets the Rollback Log, and commits all updates in the logical unit of work.

IMMEDIATE requests that a previously obtained global hold be maintained.

Note: the FRAME and END FRAME commands are QUEST language commands, but they are also available in PLEX. See the *SYSTEM 2000 PLEX Manual, Version 12, First Edition* for syntax, description, and examples.

Considerations

- A user-defined logical unit of work can cross more than one update cycle boundary.
- The SAME file is evicted when you issue a FRAME command.
- The commands DEFINE, CONTROL, REPORT, COMPOSE, and EXIT are not allowed between the FRAME and END FRAME commands. Issuing one of these commands causes the database to be damaged, a SYSTEM 2000 error message to be issued, and an exit from the software. When the next user opens the database, the database is recovered.
- Logical units of work in the Self-Contained Facility cannot cross TP segments. When Rollback is enabled for a database, any attempt to span a logical unit of work across TP segments causes the database to be damaged. For more information about TP segments, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

Abbreviations:
FRAME = FR
IMMEDIATE = IM

Chapter 23

Update Examples by Task

Index to Examples

Commands and Syntax Units	Example Numbers
ASSIGN	1
CHANGE	1 3
LIMIT	2 3
Parametric String	3
REMOVE TREE	2

Example 1

You are requested to change the address of Molly I. Gibson. Her new address is 2311 Hansford, Austin, TX 78753.

This request calls for updating three items: C14 (STREET ADDRESS), C15 (CITY-STATE), and C16 (ZIP CODE).

There are a variety of ways to do this update. They are explained below.

- The simplest way is to issue three consecutive ASSIGN commands:

```
AS C14 EQ 2311 HANSFORD* WH C3 EQ MOLLY I. AND C2 EQ GIBSON:
AS C15 EQ AUSTIN TX* WH SAME:
AS C16 EQ 78753* WH SAME:
```

  By using the ASSIGN command, the new values for the items are entered into the database regardless of whether the items are currently valued.

  Note, however, that the above commands are not equivalent to the following single ASSIGN command. This next command would enter the new values into the database, but it would also remove the other values in that C0 record.

```
AS C0 EQ 14*2311 HANSFORD*15*AUSTIN TX*16*78753*END*
   WH C3 EQ MOLLY I. AND C2 EQ GIBSON:
```

- If you know that the three items are currently valued, the following command would update the three items without removing other values from that C0 record:

```
CH C0 EQ 14*2311 HANSFORD*15*AUSTIN TX*16*78753*END*
   WH C3 EQ MOLLY I. AND C2 EQ GIBSON:
```

- If you are doing the update interactively, it might be worthwhile to issue a retrieval command such as the one below before updating the record:

```
PR C14, C15, C16 WH C3 EQ MOLLY I. AND C2 EQ GIBSON:
```

  Then you can determine which, if any, of the items must be changed and issue CHANGE or ASSIGN commands as appropriate. (For instance, perhaps only the street address needs to be changed.)

Example 2

Suppose that previous update errors have created three C200 records with the same data in the logical entry for employee number 1013. (Assume C201 equals COBOL in all three records.) You want to delete two of the records.

```
LIMIT 0,2 / TRUNCATE:

REMOVE TREE C200 WHERE C201 EQ COBOL AND C1 EQ 1013:

-340- 3 TOTAL SELECTED RECORDS -
-343- 1 RECORD(S) CANCELLED OR TRUNCATED -
-342- 2 SELECTED RECORDS -
```

The REMOVE TREE command results in three selected data records. The LIMIT command sets the maximum to two, meaning only two data records can be acted upon. Because TRUNCATE is specified instead of CANCEL (the default), the software acts on two of those records and removes them.

Example 3

For all employees who have security clearance, the last two digits of C13 (SECURITY CLEARANCE) represent the number of months remaining before renewal. At the end of each month, you need to subtract 1 from this value for those employees. You know that no more than 15 employees should qualify.

```
LIMIT 1, 15:

CHANGE C13 = (C13 -1) WHERE C13 EXISTS:

-342-       15 SELECTED RECORD(S) -
```

In this case, with the LIMIT command, you can ensure that an update affects only a predetermined subset of a database. With the LIMIT command set to a maximum of 15, only 15 records can be acted upon. If more than 15 data records are selected, the CHANGE command will not be executed.

Since this operation is to be carried out every month, the commands used can be stored in a parametric string, as shown below.

```
1001*CLEARANCE UPDATE (STRING $ LIMIT 1, *1*: CH C13 = (C13-1)
   WH C13 EXISTS: END LIMIT:$)
```

When you invoke the string, you specify the maximum limit. Therefore, the above commands can be executed simply by issuing the following:

```
*C1001 (15):
```

The 15 in parentheses replaces the first, and in this case the only, parameter in the string. The END LIMIT command cancels the limits.

Qualifying Data

The Where-Clause

Normalization

Optimization of Where-Clauses

Chapter 24

The Where-Clause

INTRODUCTION 24-2
Format 24-2
Considerations 24-5

SPECIFYING WHERE-CLAUSE CONDITIONS 24-6
Specifying the Existence of Values Using EXISTS 24-7
Specifying the Nonexistence of Values Using FAILS 24-8
Comparing an Item with a Value Using EQ, NE, GE, GT, LE, or LT 24-9
Comparing Two Items Using EQ, NE, GE, GT, LE, or LT 24-11
Comparing an Item with a Collect File Item 24-12
Collect File processing 24-13
Comparing an Item with a Range of Values Using EQ, NE, or SPANS 24-15
Searching for Characters within an Item Using CONTAINS 24-17
Considerations 24-19
Specifying the Predicate Operators IN and CTNSIN 24-23
Format 24-23
Key and Non-Key Conditions 24-24

SPECIFYING EXPRESSIONS WITH LOGICAL OPERATORS: AND, OR, NOT 24-25

DESIGNATING SPECIFIC RECORDS: HAS 24-29
Format 24-29
Considerations 24-30
Examples 24-30

SPECIFYING A POSITION AMONG SIBLINGS: AT 24-33
Format 24-34
Considerations 24-34
Examples 24-35

PROCESSING ORDER OF OPERATORS 24-37

This chapter describes the QUEST language where-clause, which you can include in retrieval and update commands to select a subset of a database.

Before you use these chapters for reference, be sure you have read Chapter 2, "Qualifying Data."

INTRODUCTION

A where-clause selects a subset of a database for retrieval and update commands.

A where-clause consists of one or more expressions following the keyword WHERE. Each expression is made up of one or more conditions that a value must meet. For example, a simple where-clause can include one expression consisting of just one condition. On the other hand, a complex where-clause can include several conditions combined into multiple expressions to qualify the desired data.

You must have W-authority (where-clause authority) for items and records specified in a where-clause.

Format

WHERE *expression*

WHERE is the keyword designating a where-clause. (In some forms of the INSERT TREE command, you can use the keyword BEFORE or AFTER instead of WHERE.)

expression

```
|condition
|(expression)
|NOT expression
|expression AND expression
|expression OR expression
|record HAS expression
|expression AT n
|SAME
```

condition

```
[NON-KEY] item  |unary-operator
                |binary-operator value
                |ternary-operator value * value
                |CONTAINS text
                |* binary-operator item *
                |* binary-operator cf-item [(CF)] *
                |IN (value, value [, value])
                |CTNSIN (value, value [, value])
```

Abbreviations and synonyms:

CONTAINS = CONT, CONTAIN, CONTAINING
EXISTS = EXIST, EXISTING
FAILS = FAIL, FAILING
HAS = HAVE, HAVING
NON-KEY = NK
SAME = SA
SPANS = SPAN, SPANNING
WHERE = WH

NOT finds the complement of specified criteria. See **Specifying Expressions with Logical Operators: AND, OR, NOT** on page 24-25.

AND combines two expressions by finding data records that satisfy both expressions. See **Specifying Expressions with Logical Operators: AND, OR, NOT** on page 24-25.

OR combines two expressions by finding data records that satisfy either expression or both. See **Specifying Expressions with Logical Operators: AND, OR, NOT** on page 24-25.

record is a schema record name or component number.

HAS specifies a focal record to qualify data records based on the contents of descendant or ancestor records. See **Designating Specific Records: HAS** on page 24-29.

AT specifies a data record by its position under its parent. See **Specifying a Position Among Siblings: AT** on page 24-33.

n is a 0 or a positive integer indicating position. Zero means last position.

SAME repeats the where-clause of a previous command with or without additional qualification criteria. See **Repeating Qualification Criteria: SAME** on page 32-5.

item is a schema item name or component number. The item can be key or non-key.

unary-operator EXISTS or FAILS.
See **Specifying the Existence of Values Using EXISTS** on page 24-7 or **Specifying the Nonexistence of Values Using FAILS** on page 24-8.

binary-operator EQ, NE, GE, GT, LE, or LT.
See **Comparing an Item with a Value Using EQ, NE, GE, GT, LE, or LT** on page 24-9, **Comparing Two Items Using EQ, NE, GE, GT, LE, or LT** on page 24-11, or **Comparing an Item with a Collect File Item** on page 24-12.

ternary-operator EQ, NE, or SPANS.
See **Comparing an Item with a Range of Values Using EQ, NE, or SPANS** on page 24-15.

value is a literal value or a system string (*NOW*, *TODAY*, *FTODAY*). Optionally, you can enclose the value with a delimiter of your choice. The default is the slash. Ternary operators require both a low value and a high value.

CONTAINS searches for characters within an item's value.

text uses the following syntax, which is explained in **Searching for Characters within an Item Using CONTAINS** on page 24-17.

```
[[THE] CHARACTERS]  substring

  [* substring [correlation]]  |[position]
                               |[interval]

[THE] |WORD     substring  |[position]
      |PREFIX               |[interval]
      |SUFFIX
```

cf-item is a Collect File item name. The item name can be followed by (CF) to resolve a conflict between a schema item name and a Collect File item name.

A Collect File item name can be used only as the right-hand term of an item-to-item comparison. Collect File items do not have component numbers associated with them. (See Chapter 10, "Creating a Collect File: COLLECT," for details about processing Collect File conditions.)

IN causes the software to create a condition for each specified value. The conditions are connected with the OR operator. Each condition contains the specified item, the EQ operator, and one of the specified values.

CTNSIN causes the software to create a condition for each specified value. The conditions are connected with the OR operator. Each condition contains the specified item, the CONTAINS operator, and one of the specified values. The software searches for the characters within the item values.

You can also use symbols for the following operators:

Operator	Symbol
EQ	=
GE	>= or => or ¬< or !<
GT	>
LE	<= or =< or ¬> or !>
LT	<
NE	¬= or !=
AND	&
OR	|
NOT	¬ or !

Without symbols, the software reads a blank on either side of an operator to separate it from other syntax units. However, with symbols, blanks are not required for separation. Any blanks found following a symbol are considered part of the value. For example, these two where-clauses are equivalent:

```
WHERE C1 EQ ABC AND C2 GT DEF:          WHERE C1=ABC&C2>DEF:
```

Considerations

- The maximum number of characters allowed for any value specified in a where-clause is 250.

- The storage required for all values in a where-clause must not exceed 368 words. To determine the number of words required to temporarily store a CHARACTER, TEXT, or UNDEFINED value, add 4 to the number of characters in the value and divide the sum by 4. The number of words is given by the integral part of the result. For example, two words are required to store the value SMITH, because (5 + 4)/4 = 2.25.

  To determine the number of words required to temporarily store an INTEGER, DECIMAL, or MONEY value, add 8 to the picture and divide the sum by 8. The number of words is given by the integral part of the result. For example, one word is required to store a value with a picture of 9999.99, because (6+8)/8 = 1.75. REAL requires one word, and DOUBLE requires two words. Dates require one word.

  If values in a where-clause exceed the 368 word limit, break up the command into smaller commands by using WHERE SAME, WHERE SAME AND, or WHERE SAME OR.

- The storage required for all operators in a where-clause must not exceed 400 words. Unary, binary, and ternary operators require 10 words each. AT, HAS, and CONTAINS use 10 words each. One word is required for each parenthesis, SAME, NOT, AND, and OR.

  If the operators in the where-clause exceed the 400-word limit, break up the command into smaller commands with fewer operators by using WHERE SAME, WHERE SAME AND, or WHERE SAME OR.

- Under Security by Entry, the first or only condition in the where-clause must be a key-condition using the EQ operator. If the condition specifying the Entry Key item is not the only condition in the where-clause, it must be joined to the other condition or expression with the AND operator. The format of a where-clause when the item is an Entry Key item is shown below.

  WHERE *item* **EQ** *value* **[AND (***expression***)]**

  See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about Security by Entry.

- If component names contain any symbols, use the component numbers. Also, if you change the system separator to be one of the symbols, syntax errors could occur if you were to specify that symbol in the where-clause.

- If a condition contains different types of numeric items, the software uses a specific order of precedence to convert the lower type to the higher type before performing the comparison. The order from highest to lowest is DOUBLE, REAL, DECIMAL, MONEY, INTEGER, and DATE.

- The software guarantees that floating point values are precisely displayed and compared; "fuzzy" logic does not creep into where-clause logic. For example, if you want to locate the value .0965E+5, you must know that the precise value was entered into the database. Otherwise, use the SPANS operator to locate a range of imprecise values.

- You can use either display or HEX notation for UNDEFINED values in where-clause conditions. Trailing binary zeros are considered insignificant. The length of an UNDEFINED item is determined by its picture specification. Use the FAILS operator to locate missing values.

- The following rules pertain when the software compares an UNDEFINED item value with a literal, a CHARACTER item, a TEXT item, or another UNDEFINED item.

  The length of the literal, CHARACTER, or TEXT item value must be equal to or shorter than the UNDEFINED item's picture. If the literal, CHARACTER, or TEXT value is longer than the undefined item picture, the values will never match.

  A short literal, CHARACTER, or TEXT value is filled with trailing binary zeros before the system performs the comparison. (TEXT item values can have trailing blanks.)

  If an UNDEFINED item value contains hexadecimal values X'00' through X'0F', the value will never match a literal, CHARACTER, or TEXT value since those item types cannot contain hexadecimal values X'00' through X'0F'.

SPECIFYING WHERE-CLAUSE CONDITIONS

A where-clause expression consists of one or more conditions. By using the operators described in this section, you specify criteria that values must meet in order for the data records containing those values to qualify for further processing.

You specify a condition by using any of the following operators:

- unary operators EXISTS and FAILS
- binary operators EQ, NE, LE, LT, GE, and GT
- ternary operators SPANS, EQ, and NE
- text search operator CONTAINS
- predicate operators IN and CTNSIN.

A data record qualifies when an item in the record meets the criteria specified in the condition. Therefore, a condition always refers to an item and not to a record.

For example, in the EMPLOYEE database, the following condition is valid because MAJOR FIELD is an item:

```
MAJOR FIELD EQ ENGLISH
```

However, the following condition is invalid because EDUCATION is a record:

```
EDUCATION EXISTS
```

Specifying the Existence of Values Using EXISTS

The EXISTS operator tests for the existence of values. EXISTS is a unary operator because it has only a single operand -- an item.

EXISTS qualifies all data records having a value for the specified item. You can use only items to test the existence of values. You cannot use records.

[NON-KEY] *item* **EXISTS**

---

item is a schema item name or component number. The item can be key or non-key.

For example,

```
PRINT LAST NAME, ACCRUED VACATION WHERE ACCRUED VACATION EXISTS:

* LAST NAME     ACCRUED VACATION
***
* GIBSON                  40.00
* REID                    80.00
* SLYE                      .00
* WATERHOUSE               8.00
* SALAZAR                 80.00
* BOWMAN                  40.00
* NATHANIEL               40.00
* GARRETT                 80.00
    .
    .
    .
```

Abbreviations and synonyms:

EXISTS = EXIST, EXISTING

NON-KEY = NK

Specifying the Nonexistence of Values Using FAILS

The FAILS operator tests for the nonexistence of values. FAILS is a unary operator because it has only a single operand -- an item.

FAILS qualifies the same data records that the expression NOT *item* EXISTS qualifies. Thus, the FAILS operator lets you specify certain criteria without using the keyword NOT.

[NON-KEY] *item* **FAILS**

---

item is a schema item name or component number. The item can be key or non-key.

For example,

```
PRINT LAST NAME WHERE ACCRUED VACATION FAILS:

2* FREEMAN
```

The previous command is equivalent to the following command:

```
PRINT LAST NAME WHERE NOT ACCRUED VACATION EXISTS:
```

Since FAILS qualifies the same records as NOT *item* EXISTS, both formats give the same results. However, there are some processing differences that are discussed in Chapter 26, "Optimization of Where-Clauses." For example, the command

```
EMPLOYEE STATUS EQ FULL TIME AND NOT ETHNIC ORIGIN EXISTS:
```

is processed faster than this command:

```
EMPLOYEE STATUS EQ FULL TIME AND ETHNIC ORIGIN FAILS:
```

Abbreviations and synonyms:
FAILS = FAIL, FAILING
NON-KEY = NK

Comparing an Item with a Value Using EQ, NE, GE, GT, LE, or LT

The EQ, NE, GE, GT, LE, and LT operators compare items with a value. These are binary operators because they use two operands: an item and a specified value.

```
[NON-KEY] item |EQ [delimiter] value [delimiter]
               |NE
               |GE
               |GT
               |LE
               |LT
```

item is a schema item name or component number. The item can be key or non-key.

value is a literal value or the system strings *NOW*, *TODAY*, or *FTODAY*.

delimiter is a special character that you can use to enclose the value. The default is the slash.

When the value includes any syntax unit, such as a reserved word, a command terminator, a system separator, OR, AT, AND, NOT, a logical symbol, or a right parenthesis, the value must be enclosed by delimiters. For example, delimiters are required in the following condition because the value contains the keyword AND:

```
SCHOOL EQ /WASHINGTON AND LEE UNIVERSITY/
```

When the item is TEXT or CHARACTER and the value is enclosed by delimiters, all characters (including blanks) found between the delimiters are included in the value. For example, if C2 is a TEXT item, both of the following expressions qualify the same data records:

```
WHERE C2 EQbbbbbABC
```

```
WHERE C2 EQb/bbbbABC/
```

But, the next two conditions do not qualify the same data records:

```
WHERE C2 EQbbbABC
```

```
WHERE C2 EQb/bABC/
```

The current delimiter must not occur in a value enclosed by delimiters.

EQ qualifies those data records containing the specified value for the item. For example, the following expression qualifies those C0 data records where the value for EMPLOYEE NUMBER is equal to 1224:

```
WHERE EMPLOYEE NUMBER EQ 1224
```

Abbreviations: NON-KEY = NK

NE qualifies those data records that contain a value for the item that is not equal to the specified value. For example, the following expression qualifies those C0 records where EMPLOYEE STATUS is not equal to FULL TIME. It does not qualify those records where EMPLOYEE STATUS is null.

```
WHERE EMPLOYEE STATUS NE FULL TIME
```

The following expression qualifies both the records with EMPLOYEE STATUS not equal to FULL TIME and those records with no values for EMPLOYEE STATUS:

```
WHERE EMPLOYEE STATUS NE FULL TIME OR EMPLOYEE STATUS FAILS
```

GE qualifies those records containing values for the item that are greater than or equal to the specified value. For example, the next expression qualifies those C0 records where the value for HIRE DATE is greater than or equal to June 1, 1989. An employee hired on or after that date would qualify.

```
WHERE HIRE DATE GE 06/01/1989
```

GT qualifies those records containing values for the item that are greater than the specified value. For example, the next expression qualifies those records where the value for HIRE DATE is greater than June 1, 1989. An employee hired on that date would not qualify, but all employees hired afterwards would qualify.

```
WHERE HIRE DATE GT 06/01/1989
```

LE qualifies those records containing values for the item that are less than or equal to the specified value. For example, the next expression qualifies those records where the value for HIRE DATE is less than or equal to June 1, 1989. An employee hired on or before that date would qualify.

```
WHERE HIRE DATE LE 06/01/1989
```

LT qualifies those records containing values for the specified item that are less than the specified value. For example, the next expression qualifies those records where the value for HIRE DATE is less than June 1, 1989. An employee hired on that date would not qualify, but all employees hired before the date would qualify.

```
WHERE HIRE DATE LT 06/01/1989
```

Comparing Two Items Using EQ, NE, GE, GT, LE, or LT

Besides comparing an item with a value, you can also use the EQ, NE, GE, GT, LE, and LT operators to compare two items.

When comparing two items, binary operators have two operands, that is, the two items to be compared. The comparison between the two operands is based on the same principles as the ones described when the operators compare an item with a value.

```
[NON-KEY] item * |EQ item *
                 |NE
                 |GE
                 |GT
                 |LE
                 |LT
```

item is a schema item name or component number. The item can be key or non-key.

* is the system separator, which must follow each specified item.

For example, the EQ operator qualifies those data records where the first item has the same value as the second item. To be eligible for comparison, both items must belong to related records and be of compatible type. Both must be either numeric or textual.

The comparison reformats the value for the second item to fit the picture of the first. When converting a DECIMAL or MONEY value for comparison to INTEGER or DATE, the software truncates digits to the right of the decimal point. (Dates are calculated as days elapsed since October 15, 1582.)

For INTEGER or DATE to DECIMAL or MONEY conversions, the implied decimal point aligns with the explicit decimal point so that 123 becomes 123.00 when compared to an item with a picture of 9999.99. Regardless of the order of the items, the qualified data records are at the same level as the lower of the records containing the specified items. For example, in the EMPLOYEE database, the following condition qualifies those C100 data records where the value of C105 equals the value for C4 in the C0 record:

```
WHERE C4 * EQ C105*
```

If a condition contains different types of numeric items, the software uses a specific order of precedence to convert the lower type to the higher type before performing the comparison. The order from highest to lowest is DOUBLE, REAL, DECIMAL, MONEY, INTEGER, and DATE.

For an item-to-item comparison of two UNDEFINED items, the shortest UNDEFINED item value is filled with trailing binary zeros before the comparison, which causes a match if the UNDEFINED value in the database has trailing binary zeros.

Comparing an Item with a Collect File Item

You can use the binary operators EQ, NE, GE, GT, LE, and LT to compare an item with a Collect File item. This type of condition is called a *Collect File condition*.

```
[NON-KEY] item * |EQ cf-item *
                 |NE
                 |GE
                 |GT
                 |LE
                 |LT
```

item is a schema item name or component number. The item can be key or non-key.

* is the system separator, which must follow each specified item.

cf-item is a Collect File item name. The item name can be followed by (CF) to resolve a conflict between a schema item name and a Collect File item name.

If the software detects a Collect File condition in a where-clause, the Collect File value in the current Collect File record is used to process the condition. If the left term of the condition is key, then the condition is processed as a key condition. If the left term is non-key (or if you make it non-key by specifying NON-KEY), the condition is processed as a non-key condition.

If you specify a Collect File item as the left term of a where-clause condition, an error message is issued, the command is ignored, and processing continues.

The manner in which the software processes a retrieval command depends upon the absence or presence of one or more Collect File items in the command's retrieval-clause and where-clause.

Collect File processing This Collect File is the example for the processing description that follows it.

	PART NUMBER	VENDOR NUMBER
record 1	101	1299
record 2	101	1323
record 3	201	-NULL-
record 4	522	1592
record 5	522	1680
record 6	629	1402
record 7	629	1892

If there are no Collect File conditions in the where-clause of a retrieval command, the where-clause is executed once, as usual. After the conditions have been processed and a single list of qualified records has been produced, this list is normalized to the schema record type of the target record, resulting in a single list of selected records. If there are no qualified records, a null list of selected records results, and no further processing is done for the command.

If the where-clause contains one or more Collect File conditions, the where-clause is executed once for each record in the Collect File. For example, if there are seven records in the Collect File, the where-clause is executed seven times, resulting in seven lists of selected records.

The Collect File is positioned at its first record, and the where-clause is executed. When a Collect File item is found, its value is retrieved and used to evaluate the where-clause condition. For example, assume for the following where-clause that VENDOR NUMBER equals 1299 in the Collect File:

```
WHERE C1* EQ VENDOR NUMBER*
```

The first execution of the where-clause for the first Collect File record is processed as

```
WHERE C1 EQ 1299
```

After the where-clause execution ends and a list of qualified records is obtained, the list is normalized to the schema record type of the target record. This results in a single list of selected records. A special end-of-list indicator is placed after this list of selected records.

The Collect File is then positioned to its next record, record 2, and the where-clause is re-executed. When a Collect File item is found, its value in the Collect File record is used. Continuing with the above example and a VENDOR NUMBER of 1323 in the second Collect File record,

```
WHERE C1* EQ VENDOR NUMBER*
```

becomes

```
WHERE C1 EQ 1323
```

After the where-clause has been executed, the list of qualified records is normalized to the schema record type of the target record, forming a list of selected records. This list of selected records is then appended after the list of selected records obtained in the first execution of the where-clause for the first Collect File record. An end-of-list indicator is placed after it, indicating the end of the second list of selected records.

This process is repeated for each of the remaining Collect File records until there are no more. After executing the where-clause once for each Collect File record, there are seven lists of selected records.

If a Collect File item is null during an execution of the where-clause, that execution is terminated and that list of selected records consists of only the end-of-list indicator. For example, if the Collect File item is null in the third record on the Collect File, the condition

```
WHERE C1* EQ VENDOR NUMBER*
```

becomes

```
WHERE C1 EQ -NULL-
```

Where-clause processing for this Collect File record is ended immediately, and the list of selected records for this execution consists only of the end-of-list indicator. That is also true when an execution of the where-clause fails to select any records.

In order to optimize the processing of a where-clause containing Collect File conditions, non-Collect File conditions are evaluated only on the first execution of the where-clause. Their results are saved for subsequent executions of the where-clause for the remaining Collect File records.

The software rearranges the where-clause, if necessary, so that the number of saved lists of non-Collect File conditions and expressions is kept to a minimum. For example, the where-clause

key-condition **AND** (*key-condition* **AND** *Collect-File-condition*)

is automatically rearranged to be processed as follows:

Collect-File-condition **AND** (*key-condition* **AND** *key-condition*)

Rearranging the conditions does not affect the final result, and the rearranged conditions are processed much more efficiently.

To summarize, upon termination of all where-clause processing, a single list of selected records is generated if the where-clause contained no Collect File items. If the where-clause did contain Collect File items, a list of selected records is generated for each Collect File record.

If you specify an ordering-clause, it is executed once for each list of selected records. However, you cannot specify Collect File items in an ordering-clause.

Comparing an Item with a Range of Values Using EQ, NE, or SPANS

The ternary operators EQ, NE, and SPANS compare an item with a range of specified values. These are called ternary operators because they have three operands: an item, a low value, and a high value.

```
[NON-KEY] item |EQ      [delimiter] low-value [delimiter] *
               |NE
               |SPANS

   [delimiter] high-value [delimiter]
```

item is a schema item name or component number. The item can be key or non-key.

delimiter is a special character with which you enclose the value. The default is the slash.

When the low value or high value includes any syntax unit (such as the command terminator, the system separator, OR, AT, AND, NOT, a logical symbol, or a right parenthesis), you must enclose the value with delimiters. The current delimiter must not occur in a value enclosed by a delimiter. For example, delimiters are required in the following condition because the values contain a right parenthesis:

```
WHERE CODE SPANS /X(1)/ * /X(12)/
```

When the item is TEXT or CHARACTER and the low value or high value is enclosed by delimiters, all characters (including blanks) found between the delimiters are included in the value. For example, the following condition qualifies those data records with values of C2 from ␣␣A to ␣B.

```
WHERE C2 SPANS␣␣␣/␣␣A/␣*␣␣/␣B/
```

low-value is a literal value or the system strings *NOW*, *TODAY*, or *FTODAY*. The low value must be less than or equal to the high value as determined by the collating sequence. See Appendix B.

* is the system separator, which separates the low and high values.

high-value is a literal value or the system strings *NOW*, *TODAY*, or *FTODAY*.

Abbreviations and synonyms:

NON-KEY = NK

SPANS = SPAN, SPANNING

EQ and SPANS qualify those data records containing a value for the item that falls within the specified range. The low value and the high value are included in the range.

Notice that EQ is both a binary and a ternary operator. Therefore, if the EQ operator is in a stored parametric string and followed by a parameter, you can supply a value or a range of values for the parameter when invoking the string.

NE qualifies those data records containing a value for the item falling outside the specified range. The low value and the high value are not included in the range. For example, the following condition qualifies data records where HIRE DATE is 01/01/88, 12/31/90, or any date in between.

```
WHERE HIRE DATE SPANS 01/01/88 * 12/31/90
```

On the other hand, the following condition qualifies data records where HIRE DATE is before 01/01/88 or after 12/31/90:

```
WHERE HIRE DATE NE 01/01/88 * 12/31/90
```

Note: the software guarantees that floating point values are precisely displayed and compared; "fuzzy" logic does not creep into where-clause logic. For example, if you want to locate the value .0965E+5, you must know that the precise value was entered into the database; otherwise, use the SPANS operator to locate a range of imprecise values.

Searching for Characters within an Item Using CONTAINS

The CONTAINS operator searches for substrings of characters within the values of a CHARACTER or TEXT item, providing a text search capability for qualification criteria.

```
[NON-KEY] item CONTAINS [[THE] CHARACTERS] substring

   [ * substring [correlation]]  |position
                                 |interval
```

```
[NON-KEY] item CONTAINS THE  |WORD    substring  |position
                             |PREFIX              |interval
                             |SUFFIX
```

item is a schema item name or component number of a CHARACTER or TEXT item. The item can be key or non-key.

substring
```
|value
|delimiter text-value delimiter
```

value is a series of characters treated as type CHARACTER or TEXT, depending on the item specified in the condition.

text-value is a series of characters treated as type TEXT; that is, all characters are used in the text search even though the item in the condition can be type CHARACTER. The maximum number of characters allowed in a substring is 250.

delimiter is a special character used to enclose the specified value. The use of the delimiter allows reserved words and symbols in values that are specified as part of the qualification criteria. The default is the slash.

Use delimiters to enclose substrings containing a keyword that could cause the software to misinterpret the command. For example, delimiters are required in the following condition because the substring includes the word IN, which is a keyword specifying position:

```
C132 EQ /CHANGE IN POSITION/
```

The current delimiter must not occur in the substring.

correlation **[WITH] MATCH [ON]** *n* **[CHARACTERS]**
Matching requires two user-supplied substrings.

Abbreviations and synonyms:
- CHARACTERS = CHARACTER, CHAR
- CONTAINS = CONTAINING, CONTAIN, CONT, CTNS
- NON-KEY = NK
- POSITION = POS
- PREFIX = PRE
- SUFFIX = SUF
- THROUGH = THRU

n an integer equal to or greater than one. The default is one.

position **IN [POSITION** *k*
[CHARACTER

k is an integer equal to or greater than one indicating a character position in a value.

interval **FROM [[POSITION]** *i* **[THROUGH [***j***]**
[[CHARACTER] **[LAST]**

i is an integer equal to or greater than one indicating the starting character position for the interval to be searched in a value. The default position is one.

j is an integer equal to or greater than *i*, indicating the ending character position for the interval to be searched in a value.

LAST indicates the last character position in a value, however long. LAST is the default.

WORD
PREFIX
SUFFIX can be used with a single substring only.

If you specify a single substring, the number of characters required for a match is determined by the number of characters in that substring. The search tests successive sets of contiguous characters in the value until a match is found, until the limit of the search is reached, or until the end of the value is found. The search is confined to those characters between the FROM and THROUGH positions or the default first through last characters.

For two substrings specifying a range, the search compares successive sets of contiguous characters in the value to see whether the set is within the specified range (inclusive). When a set of characters is found that is equal to or greater than the first specified substring, the comparison for the second substring begins at the first character in the set that matched the first substring. The matching process then continues for the length of the second substring. Testing continues as long as *n* characters remain to be compared, where *n* is the default of one or the integer specified in MATCH ON *n* CHARACTERS.

The series of characters that are searched within values is confined by the FROM and IN specifications or their defaults. FROM confines the search to the characters between *i* and *j*, inclusively. If you do not specify the interval, text search begins in position one and goes through the last (LAST) character, however long. LAST is then the rightmost character of the value. The starting position *i* must be less than or equal to the ending position *j*. If the ending position *j* is beyond the rightmost character for a value, *j* is assumed to be the last character for that value. If the beginning position *i* is beyond the rightmost character for a value, that value does not qualify.

Specifying IN *n* restricts testing to the series of characters that begins in the *n*th position. When IN is not used, text search uses all adjacent series of characters in the interval. For example, for value ABC and an interval of one through LAST. The software considers A, B, C, AB, BC, and ABC as candidates for a successful match with the specified substring(s). If you specified IN 1, only A, AB, and ABC are considered as candidates in the search.

A phantom blank is assumed to precede and follow a value if WORD is used. For PREFIX, the blank precedes the value. For SUFFIX, the blank follows the value. The blank is temporarily associated with the value for matching purposes but is not actually present in the value.

With WORD, the series of adjacent characters tested must be preceded and followed by a phantom blank, a blank, or a special character, that is, any character less than hexadecimal 81, the lowercase a. That special character must immediately precede (or follow) the series of characters. For example, if the value is CAT␣ AND␣ DOG, search candidates are the words CAT, AND, and DOG, if WORD is used. Likewise, for the value -CAT-NO., the search candidates are CAT and NO without the hyphens and period.[1] With PREFIX, the rules are the same as for WORD except that the blank or punctuation character need only precede the series of characters in the value. Similarly, for SUFFIX, the blank or special character need only follow the series of characters in the value.

The MATCH option allows you to specify the minimum number of characters that must match in the range of the two specified substrings. For instance, MATCH ON 3 with substrings ABC*BCK indicates that you want a match for ABC, ABD, ABE, and so on, through BCK, but not for the single character B or for AC, AD, and so on, or BC. If MATCH is not specified, the default minimum is one.

Considerations

- You cannot use the text search capability with *item separator* CONTAINS.

- In order for the substring (or range given by two substrings) to produce a qualified data record, it must be completely contained within the interval specified by *i* and *j*. If (*i* + *n* -1) is greater than *j*, the command is rejected. When a single substring is given, *n* equals the number of characters in the substring. For a range of two substrings, the value of *n* in MATCH ON *n* CHARACTERS is used or the default of one.

- Text search for a key item means searching the index (one match for each existing value) for the substring(s). Therefore, if possible, the where-clause should be constructed with one or more standard key conditions (no text search) combined with non-key text search conditions. Such a construction would confine text searches to isolated portions of the database that were qualified through use of key item conditions and the index.

  For example, if C1 and C2 are key items, use WHERE C1 LT 1234 AND NK C2 CONTAINS MAC. This where-clause isolates those entries having C1 less than 1234. Then those data records are tested in non-key mode to see whether C2 contains the substring MAC. If C2 had not been specified as non-key, all values in the index for C2 would have been tested by text search before the AND operation could have been processed for the two key conditions.

---

[1] Using delimiters makes more search candidates possible, such as /CAT AND/, /CAT AND DOG/ or /-CAT-NO./, /NO./, and so on.

Example 1

Given the four CHARACTER values and the various user-specified substrings, the text search capability produces the results shown below. The example lists the values that qualify (by number) and the specific character sequences that caused a successful match. The $ character is the current delimiter (DELIMITER IS $:).

CHARACTER Values

```
1) QM6ƀPL5ƀRM          3) PQLƀRM358<2PM
2) RMƀ302-LA           4) A-ƀBC
```

User Specifications	Qualifying Values
PREFIX RM3	3) = RM3
K4*L99	1) = L5ƀ,. 2) = LA, 3) = LƀR
K4*L99 MATCH 3	1) = L5ƀ, 3) = LƀR
Mƀ1*M5	2) = Mƀ3, 3) = M35
CHAR PB*PW	1) = PL, 3) = PQ
PREFIX RM	1) = RM, 2) = RM, 3) = RM
$ƀRM$	1) = ƀRM, 3) = ƀRM
PREFIX $ƀAB$	4) = ƀAB
PREFIX $ƀRM$	None
A*AB	2) = A, 4) = A
299 * 301 MATCH 2 FROM 2 THROUGH 5	2) = 30
300 * 302 MATCH 2 FROM 2 THROUGH 5	None

Example 2

This example shows the results of text search on four TEXT values and indicates the success or failure of each user specification.

TEXT Values

1) ♪ABC
2) ABC♪E
3) QABC
4) ♪ABCD

User Specification	Success or Failure
ABC	1), 2), 3), 4)
ABC IN 1	2)
ABC FROM 1 THRU 3	2)
ABC FROM 1 THRU 2	-397- SEARCH RANGE INVALID -
ABC FROM 1 THRU LAST	1), 2), 3), 4)
PREFIX ABC	1), 2), 4)
PREFIX ABC IN 1	2)
PREFIX ABC FROM 1 THRU 2	-397- SEARCH RANGE INVALID -
PREFIX ABC FROM 1 THRU 3	2)
PREFIX ABC FROM 1 THRU 4	1), 2), 4)
PREFIX ABC FROM 2 THRU 4	1), 4)
PREFIX ABC FROM 1 THRU LAST	1), 2), 4)
WORD ABC IN 1	2)
WORD ABC FROM 2 THRU LAST	1)
SUFFIX ABC	1), 2), 3)
ABC IN 1 THRU 3	-100- SYNTAX ERROR -

Example 3

Using the six hypothetical CHARACTER values and the substrings and options shown, this example shows the use of two user-supplied substrings (range test) with optional MATCH and IN specifications.

CHARACTER Values

```
1)  ABCD                    4)  ABCDE
2)  ABXX                    5)  AC
3)  AB                      6)  B
```

User Specifications	Success or Failure
WORD ABC*D	-368- TOO MANY OPERANDS FOR OPERATOR - 1), 2), 4)
CHARACTER ABC*D MATCH 4	
PREFIX ABC*D	-368- TOO MANY OPERANDS FOR OPERATOR -
SUFFIX ABC*D MATCH 4	-368- TOO MANY OPERANDS FOR OPERATOR -
WORD ABC*D	-368- TOO MANY OPERANDS FOR OPERATOR - 1), 2), 3), 4), 5), 6)
CHARACTER ABC*D	
CHARACTER ABC*D MATCH 5	4)
CHAR ABC MATCH 2	-100- SYNTAX ERROR -
CHAR ABC*D IN 1	1), 2), 4), 5), 6)

Specifying the Predicate Operators IN and CTNSIN

The IN and CTNSIN operators cause the software to create a condition for each specified value and to connect the conditions with the OR operator. IN and CTNSIN are sometimes called predicate operators. That is, the predicate operators give you an abbreviated format for specifying two or more conditions joined by the OR operator.

Format

```
item |IN      (value, value, [, value...])
     |CTNSIN
```

item is a schema item name or component number. The item can be key or non-key.

value is a data value that will be used in the condition created by the software.

For the IN operator, each condition contains the specified item, the EQ operator, and one of the specified values. You cannot give a range of values; that is, each specified value is treated as a single data value. Here is an example of the IN operator, followed by the expressions that the software constructs and processes:

```
WHERE C2 IN (SMITH, JONES, GREEN)

WHERE (C2 EQ SMITH OR C2 EQ JONES OR C2 EQ GREEN)
```

For the CTNSIN operator, each condition contains the specified item, the CONTAINS operator, and one of the specified values. The software searches for each specified value as a substring within the item's values. Options such as PREFIX, SUFFIX, WORD, and MATCH are not allowed. Here is an example of the CTNSIN operator, followed by the expressions that the software constructs and processes:

```
WHERE C2 CTNSIN (SER, LER, TER)

WHERE (C2 CONTAINS SER OR C2 CONTAINS LER
  OR C2 CONTAINS TER)
```

The HAS, AT, and AND operators apply to all conditions generated for IN and CTNSIN. Here is an example of an IN condition with the AND operator, followed by the expression that the software constructs and processes:

```
WHERE SKILL TYPE IN (COBOL, FORTRAN)
  AND C1 LT 1050:

WHERE (SKILL TYPE EQ COBOL OR SKILL TYPE EQ FORTRAN)
  AND C1 LT 1050:
```

This example illustrates HAS with CTNSIN and the resulting where-clause:

```
WHERE CO HAS C412 CTNSIN (MA, PH)

WHERE CO HAS (C412 CONTAINS MA OR C412 CONTAINS PH)
```

Key and Non-Key Conditions

Conditions are either *key* or *non-key*. Key conditions use the index to produce lists of qualified data records. Non-key conditions require a search through every entry in the Data Table for records that are occurrences of the schema record containing the specified item.

Key conditions must include a key item. Non-key conditions specify non-key items, the FAILS operator, or two items for comparison. Key conditions can become non-key conditions if you specify the keyword NON-KEY. The following illustration summarizes this information.

Illus. 24.1 Key-Conditions and Non-Key-Conditions

Condition	Treated as a Key-Condition	Treated as a Non-Key-Condition
SAME	always	never
item EXISTS	item is key	item is non-key
item FAILS	never	always
item |EQ *value* |NE |GE |GT |LE |LT	item is key	item is non-key
item |EQ *value* * *value* |NE |SPANS	item is key	item is non-key
item |EQ |*item* * |NE |*cf-item* * |GE |GT |LE |LT	if left item is key and right item is a Collect File item.	if both items are database items, or if left item is non-key and right item is a Collect File item
item CONTAINS *text*	item is key	item is non-key
IN (*value, value*) CTNSIN (*value, value*)	item is key	item is non-key
NON-KEY *condition*	never	always

SPECIFYING EXPRESSIONS WITH LOGICAL OPERATORS: AND, OR, NOT

You can combine or modify expressions in a where-clause by using the logical operators AND, OR, and NOT. The manner in which the software processes expressions using these operators is described in **Constructing Expressions** on page 26. This section describes the syntax for constructing expressions that include these operators.

The AND operator combines two expressions by finding those data records that satisfy both expressions. You can use the AND operator only when the schema record of the qualified data records for one expression is related to the schema record of the qualified data records for the other expression.

If the schema record in both expressions is the same, the resulting list of qualified data records consists of those records appearing in both lists. If one schema record is an ancestor of the other, the qualified data records for the ancestor schema record are replaced by those descendant records that are similar to the other schema record. The resulting list of qualified data records consists of the records appearing in both lists.

The OR operator combines two expressions by finding those data records that satisfy either or both expressions. You can use the OR operator only when the schema record of the qualified data records for one expression is related to the schema record of the qualified data records for the other expression. The predicate operators IN and CTNSIN imply several conditions connected with the OR operator (see **Specifying the Predicate Operators IN and CTNSIN** on page 24-23).

If the schema record of both expressions is the same, the resulting list of qualified data records consists of those records that are found in either list; that is, the lists are joined and duplicates are discarded. If one schema record is an ancestor of the other, the qualified data records for the ancestor schema record are replaced by those descendant records that are similar to the other schema record, or in cases when such a record does not exist, by the lowest possible ancestor of that record if it existed. The resulting list of qualified records consists of those records that are found in either list.

The NOT operator modifies an expression by replacing the list of qualified records for the expression with a new list of records. The new list contains those records that are similar to the data records on the original list but are not on it, as well as higher level data records that could have descendants similar to the qualified records but do not.

When a where-clause expression includes several logical operators, the processing order is as follows:

- The NOT operator is processed first.
- Then the expressions joined by AND are processed.
- Finally, the expressions joined by OR are processed.

You can further control where-clause processing by nesting expressions, that is, enclosing an expression in parentheses. Nested expressions are processed before unnested expressions. For example, the following expression would qualify male employees who majored in mathematics, yielding unanticipated results:

```
MAJOR FIELD EQ MATH OR MINOR FIELD EQ MATH AND GENDER EQ FEMALE
```

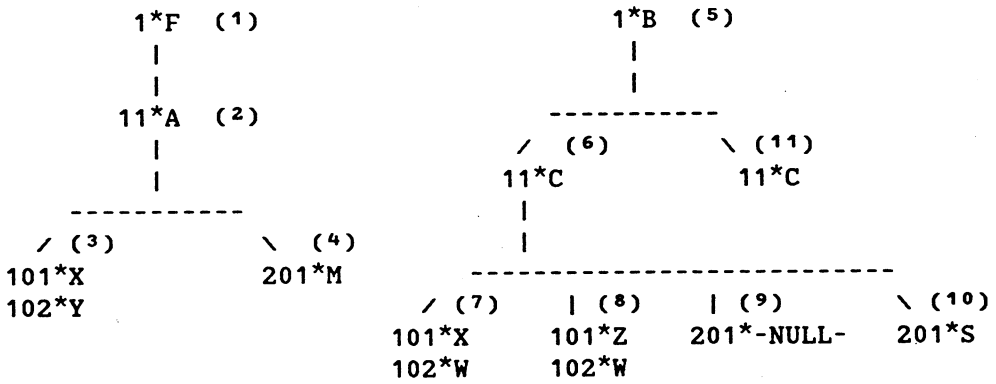
On the other hand, the following expression qualifies only female employees who either majored or minored in mathematics:

```
(MAJOR FIELD EQ MATH OR MINOR FIELD EQ MATH) AND GENDER EQ FEMALE
```

(A more detailed discussion of processing order for operators is found in **Processing Order of Operators** on page 24-37.)

If the conditions in a where-clause related by AND or OR contain components that are not in the same schema path (disjoint data records), you must use the HAS operator to raise all data records satisfying the conditions to at least the level of the first intersection of the paths. If the HAS operator is not used with the disjoint data records, the request is not processed and an error message is issued. (See **Designating Specific Records: HAS** on page 24-29 for use of the HAS operator.)

The following examples use a database consisting of the two data trees shown below. The number next to each data record represents the Hierarchical Table address of the record.



Example 1

```
C101 EQ X AND C102 EQ W
```

- Qualified records for C101 EQ X: 3, 7.
- Qualified records for C102 EQ W: 7, 8.
- Qualified record for C101 EQ X AND C102 EQ W: 7.

Example 2

```
C101 EQ X AND C11 EQ C
```

- Qualified records for C101 EQ X: 3, 7.
- Qualified records for C11 EQ C are records 6 and 10. These are replaced by those descendants that are similar to record C10: 7, 8.
- Qualified record for C101 EQ X AND C11 EQ C: 7.

Example 3

```
C101 EQ X OR C102 EQ W
```

- Qualified records for C101 EQ X: 3, 7.
- Qualified records for C102 EQ W: 7, 8.
- Qualified records for C101 EQ X OR C102 EQ W: 3, 7, 8.

Example 4

```
C101 EQ X OR C1 EQ B
```

- Qualified records for C101 EQ X: 3, 7.
- Qualified record for C1 EQ B is record 5. This record is replaced, where possible, by those descendants that are similar to record C100: 7, 8, 10.
- Qualified records for C101 EQ X OR C1 EQ B: 3, 7, 8, 10.

Example 5

```
NOT C201 EQ M
```

- Qualified record for C201 EQ M: 4.
- Qualified records for NOT C201 EQ M: 9, 10, 11.
- Records 9 and 11 are similar records not in the original list. Record 10 is a record that could have C200 descendants, but does not.

DESIGNATING SPECIFIC RECORDS: HAS

The HAS operator allows you to specify a focal record for a where-clause expression. HAS modifies an expression by replacing the list of qualified data records for the expression to the right of the HAS by a new list. This new list consists of those data records' ancestors or descendants that are occurrences of the schema record specified to the left of the HAS operator.

In the EMPLOYEE database, the HAS operator in the following expression causes the qualified records for the condition C201 EQ COBOL to be replaced by their parent C0 records:

```
C0 HAS C201 EQ COBOL
```

Use HAS to specify

- a focal record when two expressions are joined by AND or OR and the qualified data records occur in disjoint schema records. See Example 1.
- a focal record when two conditions are joined by the AND operator and the same schema item is specified in both conditions. See Example 2.
- a focal record when the target record in the action-clause is disjoint from the record qualified by the where-clause. In the case of retrieval commands, you can use a by-clause to specify a focal record for retrievals, thereby relating the qualified records to the target record. See Example 3.
- logical placement of a record in combination with the AT operator. See Example 4.

Format

record **HAS** *expression*

record is a schema record name or component number.

expression See **Introduction** on page 24-2.

Synonyms: HAS = HAVE, HAVING

Considerations

- The NON-KEY or NK indicator cannot precede the record specified to the left of HAS.
- The HAS operator cannot be nested more than 14 levels in an expression. For example, the following expression cannot be processed if C1500 is at the 15th level:

```
WHERE C10 HAS (C20 HAS (...C1400 HAS (C1500 HAS C1601 EQ A)...))
```

Examples

The following examples use the EMPLOYEE database.

Example 1

The HAS operators in the following expression are used to specify C0 as a focal record, since the conditions refer to items in disjoint schema records. C120 and C200 are disjoint records. In this case, the qualified records for the expression are C0 records.

```
WHERE C0 HAS C121 EQ 05/31/1991 AND C0 HAS C201 EQ COBOL
```

It is not necessary to specify the HAS operator in front of both expressions. The choice depends on what type of record is to be included in the list of qualified records for the expressions.

For example, the qualified records for this expression are C200 records:

```
WHERE C0 HAS C121 EQ 05/31/1991 AND C201 EQ COBOL
```

And the qualified records for this expression are C120 records:

```
WHERE C121 EQ 05/31/1991 AND C0 HAS C201 EQ COBOL
```

In any case, the HAS operator must be specified either before one of the expressions or before each expression. Otherwise, if the HAS operator is not specified, as for example, in the following expression, the software issues a message, and the command is not processed:

```
WHERE C121 EQ 05/31/1991 AND C201 EQ COBOL

337- DISJOINT CONDITIONS ENCOUNTERED-
```

The expressions shown in this example use the AND operator to combine the conditions; however, the same considerations apply for the OR operator.

Example 2

The HAS operators in this expression specify C0 as the focal record since the conditions refer to the same schema item (C201). In this case, the qualified data records are C0 records. As in Example 1, the HAS operator need not be specified in front of each condition, although it is more efficient to do so.

```
WHERE C0 HAS C201 EQ COBOL AND C0 HAS C201 EQ FORTRAN
```

If the HAS operator is not specified, no records will qualify, because there can never be a data record that has both values for the same item.

Example 3

The HAS operator in this command specifies C0 as the focal record. By doing so, the C120 data records that qualify for the condition C121 EQ 05/31/1991 are replaced by their parent C0 records, which are related to the C200 target record.

```
PRINT C201 WHERE C0 HAS C121 EQ 05/31/1991:
```

The same effect can be achieved by specifying a by-clause in the retrieval-clause. For example:

```
PRINT BY C0, C201 WHERE C121 EQ 05/31/1991:
```

Notice that the software cannot process the following command because the target record (C200) is disjoint from the type of record (C120) qualified by the condition in the where-clause:

```
PRINT C201 WHERE C121 EQ 05/31/1991:

-387- ACTION-CLAUSE COMPONENTS MUST BE IN WHERE-CLAUSE PATH-
```

Therefore, when the target record and the type of record that qualifies for the where-clause are disjoint, either specify HAS in the where-clause or specify a by-clause in the retrieval-clause.

Example 4

When a database is maintained so that the most recent occurrence of a data record is first or last under its parent, HAS combined with an AT operator can be used as shown here.

Consider a logical entry in the EMPLOYEE database in which the most recent C100 (POSITION WITHIN COMPANY) record is leftmost (first) under its parent. To obtain this most recent record for some employee, you could use the command

```
PRINT C100 WHERE (C100 HAS C1 EQ 1120) AT 1:
```

instead of the command

```
PRINT C100 WHERE C101 EXISTS AT 1 AND C1 EQ 1120:
```

In this situation, the expression using HAS would process a little faster than the expression using AND. (Note the use of parentheses to force the HAS processing to precede the AT processing.)

Notice also that the focal records (C100) are below the level of the records qualified by the condition C1 EQ 1120. In this example, the HAS operator lowered the level of the qualified records.

SPECIFYING A POSITION AMONG SIBLINGS: AT

The AT operator allows you to specify a data record by its position under its parent relative to its similar siblings. Including the AT operator in an expression means that to qualify, a data record must not only satisfy the expression but also occupy a specific position under its parent and among its similar siblings.

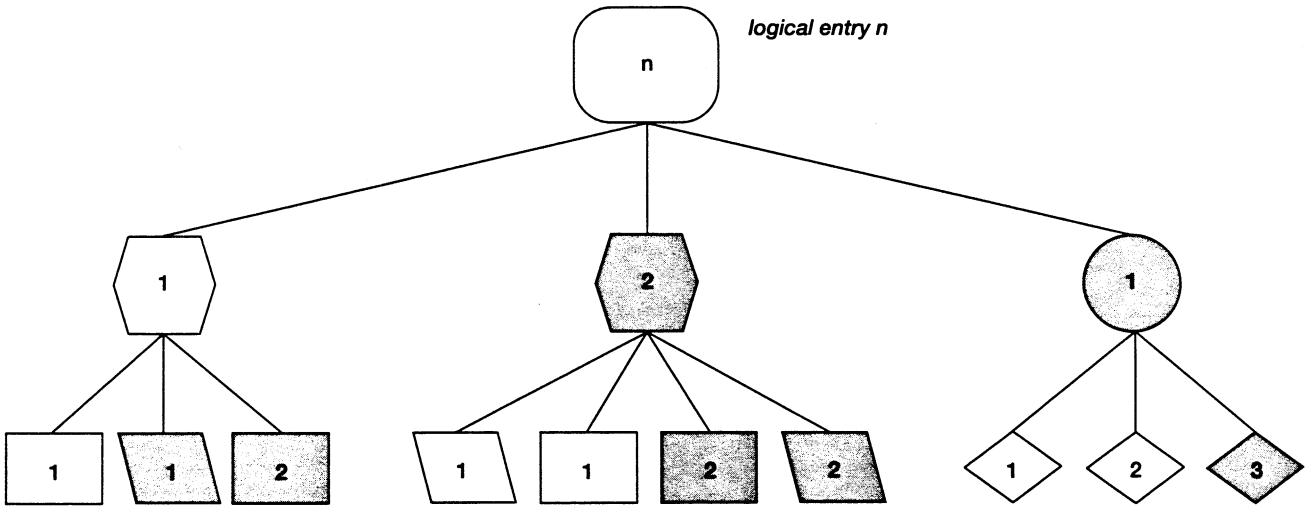
The position of a data record is simply its number in a left-to-right enumeration of similar siblings. That is, counting from left to right, if a data record is the *n*th of its kind among its siblings, then its position under its parent relative to its similar siblings is *n*.

For a data record at level 0, its position is *n* if it is the *n*th logical entry.

If a data record is the last one among its similar siblings, then it is said to be in position zero. Thus, a data record can be said to have two positions if it is the last data record of that schema record under its parent: one position is zero (last) and the other is a non-zero positive integer. No data record can have two non-zero positions or more than two positions.

The following illustration shows a logical entry with the relative positions of all the data records. Shaded records indicate the data records at position zero.

Illus. 24.2 Relative Position under a Parent



The AT operator must follow an expression and be followed by a number (positive integer or zero). AT operates on the list of qualified records for the expression in the following manner:

> For each data record in the list, the software tests whether the record's relative position among its similar siblings is the same as the number specified to the right of AT. If the test is successful, then the record is placed on the resulting list; otherwise, it is discarded. In case the original list contains records from different levels, AT always operates on the data records at the lowest level. Data records at higher levels are discarded. Thus, the resulting list of qualified records contains data records from the same schema record and having the same relative position under their parents.

Format

expression AT *n* [AT 0]

expression See **Introduction** on page 24-2.

n is a 0 or a positive integer indicating position. Zero means last position.

Considerations

- If two AT specifications are given, one must be AT 0. AT 2 AT 3 is meaningless; it requests data records that are simultaneously in the second and the third position under their parents. AT 2 AT 0 is acceptable; it requests data records that are in the second and the last position under their parents.

- You can include more than one AT specification if the specifications are separated by parentheses. For example, the use of the AT operators in the following command is valid. The command retrieves the most current value for C126 for the employee with C1 equaling 1120.

```
PRINT C126 WHERE
   (C120 HAS (C110 HAS (C100 HAS C1 EQ 1120) AT 1)AT 1) AT 1:
```

- Because of the way pointers are set in the Hierarchical Table, it is much faster to process AT 1 than to process AT 0 or AT *a-large-number*, especially when there is a large number of siblings. However, at level 0, AT 0 can be processed as fast as AT 1.

- The number specified after the AT operator must be zero or a positive integer less than or equal to 2^{24}.

Examples

The following examples show uses of the AT operator. The examples use the EMPLOYEE database where the data records for C100, C110, and C120 are maintained in order of the most recent first; that is, the most recent record is always inserted in first position.

Example 1

The following command displays the DEPARTMENTs (C102) and their current MANAGERs (C103).

```
LIST C102, C103 WHERE C102 EXISTS AT 1:

* DEPARTMENT       MANAGER
***
* INFORMATION SY   JC
  STEMS
* INFORMATION SY   MYJ
  STEMS
* ADMINISTRATION   SQT
   & FINANCE
* INFORMATION SY   MYJ
  STEMS
* EXECUTIVE
* EXECUTIVE        CPW
* EXECUTIVE        CPW
* EXECUTIVE        HEB
* EXECUTIVE        HEB
* EXECUTIVE        OMG
* MARKETING        OMG
    .
    .
    .
```

Example 2

The following command displays the EMPLOYEE NUMBER (C1) and the second value of POSITION TITLE (C101) of all employees who have held at least two positions.

```
LIST C1, C101 WHERE C101 EXISTS AT 2:

* EMPLOYEE NUMBER   POSITION TITLE
***
*            1120   ASSISTANT PROG
                    RAMMER
*            1003   SECRETARY
*            1006   MANAGER OF SAL
                    ES
*            1015   JR SALES REPRE
                    SENTATIVE
*            1145   JR SALES REPRE
                    SENTATIVE
*            1050   JR SALES REPRE
                    SENTATIVE
```

Example 3

The following command displays the DEPARTMENT (C102) where each employee started work. The EMPLOYEE NUMBER (C1) is also displayed.

```
LIST C1, C102 WHERE C101 EXISTS AT 0:

* EMPLOYEE NUMBER   DEPARTMENT
***
*            1043   INFORMATION SY
                    STEMS
*            1120   INFORMATION SY
                    STEMS
*            1265   ADMINISTRATION
                     & FINANCE
*                   INFORMATION SY
                    STEMS
*            1001   EXECUTIVE
*            1003   EXECUTIVE
*            1002   EXECUTIVE
*            1101   EXECUTIVE
*            1006   MARKETING
*            1004   EXECUTIVE
*            1007   MARKETING
*            1062   MARKETING
*            1015   MARKETING
*            1327   MARKETING
*            1145   MARKETING
    .
    .
    .
```

Example 4

The following command displays the EMPLOYEE NUMBER (C1) of all employees who have held only two positions since they were hired.

```
LIST C1 WHERE C101 EXISTS AT 2 AT 0:

* EMPLOYEE NUMBER
***
*            1120
*            1003
*            1015
*            1145
*            1050
```

PROCESSING ORDER OF OPERATORS

When you use operators to combine or modify conditions, the software processes the operators in a specific order. Understanding the processing order is important in constructing where-clauses that reflect the qualification criteria that you have in mind. The processing order of operators is as follows:

1. AT
2. HAS
3. NOT
4. AND
5. OR

That is, if an expression includes all these operators, and there are no parentheses altering the processing order, AT is processed first, then HAS, followed by NOT, AND, and finally OR.

You can further control the processing order by nesting conditions in parentheses. The conditions within the innermost set of parentheses are processed first, followed by the next deepest, moving outward until all parentheses have been processed.

When constructing a where-clause, you may want to consider the effect that the placement of non-key-conditions, key-conditions, and parentheses can have on processing time. For example, if A, B, and C are non-key-conditions and X is a key-condition, both the following expressions qualify the same data records. However, the second expression takes less time to process.

```
(A AND (B AND (C AND X)))

A AND B AND C AND X
```

See **Factors Affecting the Efficiency of Where-Clause Processing** on page 26-5 for a discussion of efficiency considerations in where-clause processing.

The following example shows the order of processing a condition preceded by a HAS operator and followed by an AT operator.

Example 1

```
C0 HAS C21 EXISTS AT 2
```

Step 1: evaluate C21 EXISTS.

Step 2: the results of Step 1 **AT** 2.

Step 3: C0 **HAS** the results of Step 2.

In the following examples, conditions are represented by the letters A, B, and C. The next three examples show the processing order of the HAS, AND, and OR operators. Notice how the placement of parentheses affects the results.

Example 2

```
record HAS A AND B OR C
```

Step 1: record **HAS** A.

Step 2: the results of Step 1 **AND** B.

Step 3: the results of Step 2 **OR** C.

Example 3

```
record HAS A AND (B OR C)
```

Step 1: record **HAS** A.

Step 2: **(**B OR C**)**.

Step 3: the results of Step 1 **AND** the results of Step 2.

Example 4

```
record HAS (A AND B OR C)
```

Step 1: A **AND** B.

Step 2: the results of Step 1 **OR** C.

Step 3: record **HAS** the results of Step 2.

The next four examples show the processing order of the HAS, AND, and AT operators. Notice how the placement of parentheses affects the results.

Example 5

record HAS A AND B AT *n*

Step 1: record **HAS** A.

Step 2: B **AT** n.

Step 3: the results of Step 1 **AND** the results of Step 2.

Example 6

record HAS (A AND B) AT *n*

Step 1: **(**A AND B**)**.

Step 2: the results of Step 1 **AT** n.

Step 3: record **HAS** the results of Step 2.

Example 7

(*record* HAS A AND B) AT *n*

Step 1: record HAS A.

Step 2: the results of Step 1 **AND** B.

Step 3: the results of Step 2 **AT** n.

Example 8

A AND *record* HAS B AT *n*

Step 1: B **AT** n.

Step 2: record **HAS** the results of Step 1.

Step 3: A **AND** the results of Step 2.

The last two examples show the processing order of the AND, NOT, and OR operators. Notice how the placement of parentheses affects the results.

Example 9

```
NOT A AND B OR C
```

Step 1: **NOT** A.

Step 2: the results of Step 1 **AND** B.

Step 3: the results of Step 2 **OR** C.

Example 10

```
NOT (A AND B) OR C
```

Step 1: **(**A AND B**)**.

Step 2: **NOT** the results of Step 1.

Step 3: the results of Step 2 **OR** C.

Chapter 25

Normalizing with the HAS Operator

Normalization is the process by which the software obtains ancestors or descendants of qualified data records during where-clause processing. Normalization can occur in two situations:

- when your where-clause contains conditions specifying items from different schema records
- when you are retrieving or updating from records that are ancestors or descendants of the qualified data records.

When you are accessing data, any record being accessed must be in the family of a qualified record. Sometimes you need the HAS operator to achieve this.

When two conditions are joined by the AND operator, there must be a way of comparing the two lists of records generated by the conditions. The HAS operator is often required for this comparison to occur and be meaningful.

When two conditions are joined with either AND or OR, four cases are possible. These cases are discussed here.

- **Case 1**
  Each condition specifies different items in the same schema record. This is the case of the examples shown for the AND and OR operators in **Constructing Expressions** on page 26. The item SKILL TYPE in the condition

  ```
  SKILL TYPE EQ COBOL
  ```

  and the item YEARS OF EXPERIENCE in the condition

  ```
  YEARS OF EXPERIENCE EQ 4
  ```

  are both in the same schema record, namely, the record JOB SKILLS.

  Notice that normalization is not required to process the following expressions:

  ```
  SKILL TYPE EQ COBOL AND YEARS OF EXPERIENCE EQ 4

  SKILL TYPE EQ COBOL OR YEARS OF EXPERIENCE EQ 4
  ```

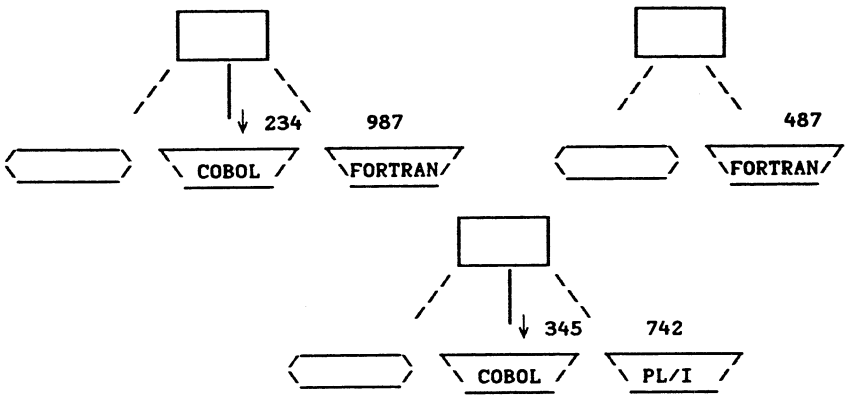
- **Case 2**
  Each condition specifies the same item. For example, suppose you want to find those employees that know both COBOL and FORTRAN. At first glance, it seems that the expression

  ```
  SKILL TYPE EQ COBOL AND SKILL TYPE EQ FORTRAN
  ```

represents the above criteria. However, as shown next, this expression does not yield the desired results.

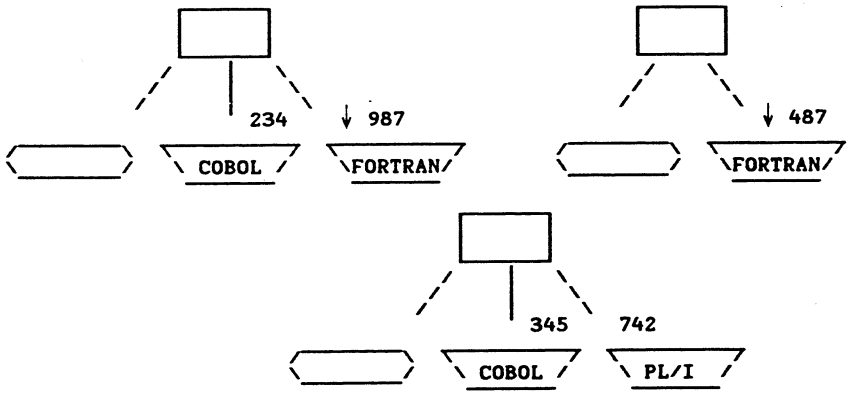
The following example shows three data trees. The numbers above the figures represent the Hierarchical Table addresses of some of the records. Arrows point to the qualified records for the following condition:

```
SKILL TYPE EQ COBOL
```



Arrows point to the qualified records for the following condition:

```
SKILL TYPE EQ FORTRAN
```



To process the following expression, the software finds those records that are common to the two lists generated by the conditions, that is, those record addresses that appear in both of the lists below:

```
SKILL TYPE EQ COBOL AND SKILL TYPE EQ FORTRAN
```

SKILL TYPE EQ COBOL *234 and 345*
SKILL TYPE EQ FORTRAN *487 and 987*

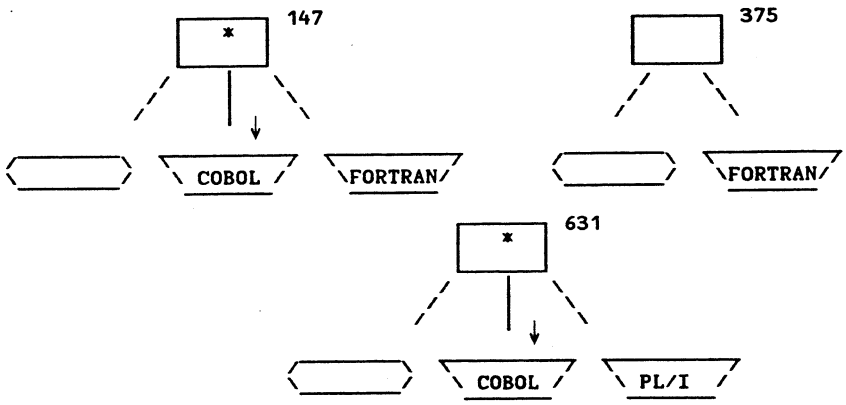
Since there are no numbers that appear in both lists, the software concludes that there are no records to qualify for the expression. However, you are not interested in the records that have both values; there is no such record. You want the employees that have both skills, in other words, those employees whose logical entries contain a

record with a value of COBOL for SKILL TYPE and also a record with a value of FORTRAN for the same item. This means, find the ENTRY records that have a descendant record with SKILL TYPE equal to COBOL and also a descendant record with SKILL TYPE equal to FORTRAN.

```
ENTRY HAS SKILL TYPE EQ COBOL AND ENTRY HAS SKILL TYPE EQ FORTRAN
```

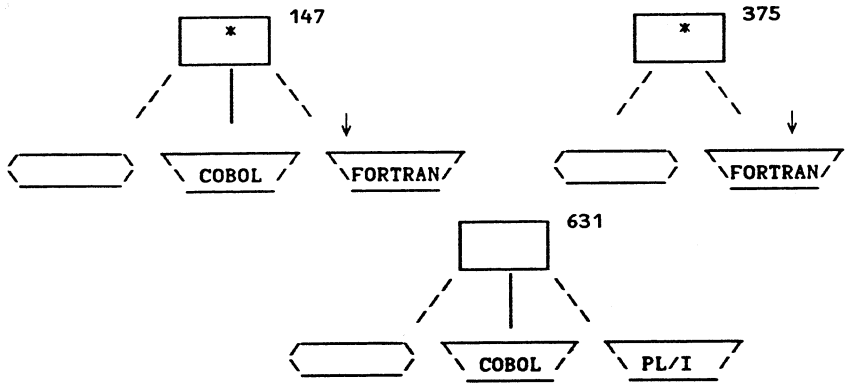
Asterisks indicate the ENTRY records that qualify for the following condition:

```
ENTRY HAS SKILL TYPE EQ COBOL
```



Asterisks indicate the ENTRY records that qualify for the following condition:

```
ENTRY HAS SKILL TYPE EQ FORTRAN
```



To process the following expression, the software first processes the HAS operator and then processes the logical operator AND that joins the two expressions with HAS.

```
ENTRY HAS SKILL TYPE EQ COBOL AND ENTRY HAS SKILL TYPE EQ FORTRAN
```

The software finds those ENTRY records that appear in both lists, that is, those numbers that appear in both of the lists below:

ENTRY HAS SKILL TYPE EQ COBOL 147 and 631
ENTRY HAS SKILL TYPE EQ FORTRAN 147 and 375

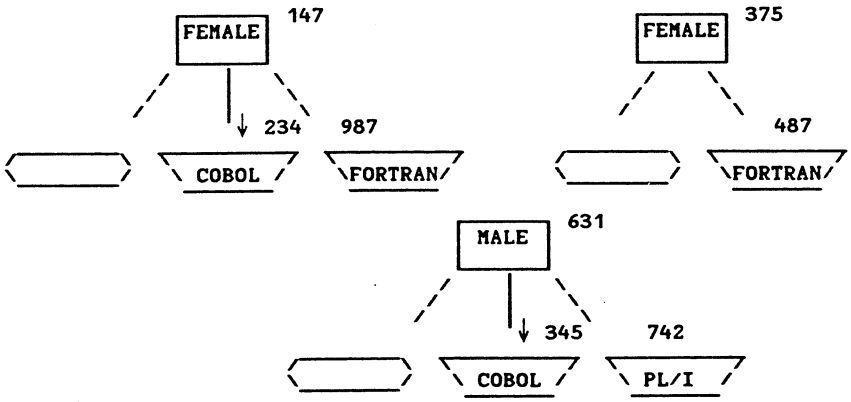
Since record 147 is found in both lists, the ENTRY record with that address qualifies for the expression. Notice that normalization is needed to obtain the desired result.

Notice also that to process the following OR criteria, normalization is not required. The records that qualify are the ones with addresses 234, 345, 487, and 987.

```
SKILL TYPE EQ COBOL OR SKILL TYPE EQ FORTRAN
```

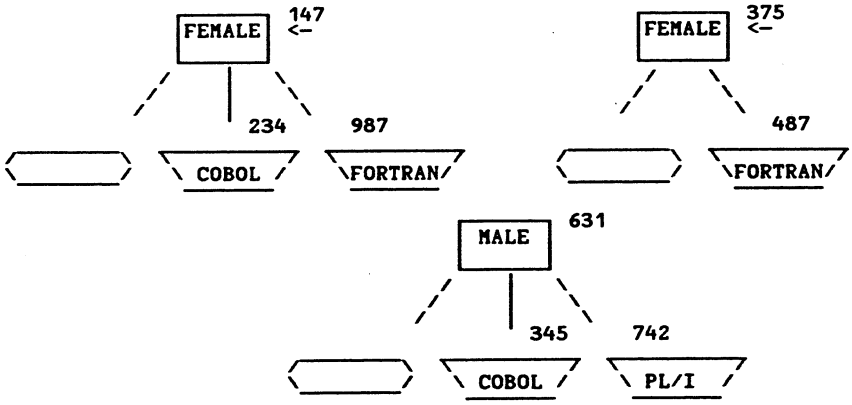
- **Case 3**
  Each condition specifies items from related schema records. The following examples show how the software processes expressions consisting of two conditions joined by AND or OR, when each condition specifies items from related records. The example consists of three data trees. The numbers represent Hierarchical Table addresses. Arrows point to the data records that qualify for this condition:

```
SKILL TYPE EQ COBOL
```



In the following example, arrows point to the data records that qualify for this condition:

```
GENDER EQ FEMALE
```



Now, when the software processes the expression

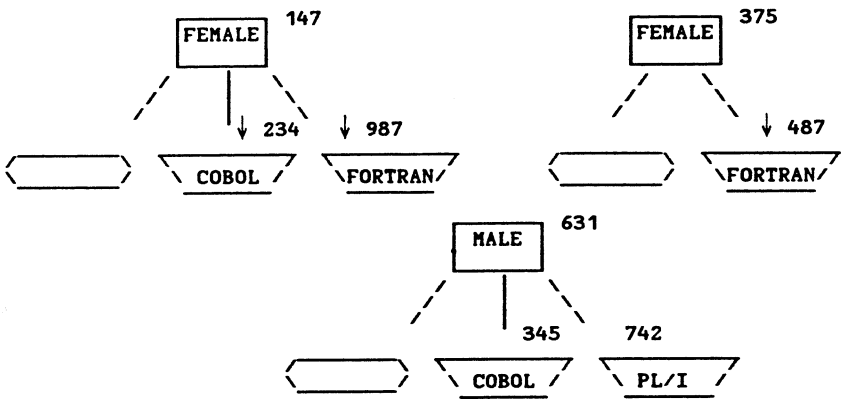
```
SKILL TYPE EQ COBOL AND GENDER EQ FEMALE
```

or the expression

```
SKILL TYPE EQ COBOL OR GENDER EQ FEMALE
```

it generates the lists of qualified records for each condition. These two lists, however, do not contain similar records. Therefore, comparing the two lists is impossible unless some prior adjustment takes place. In order to evaluate either expression, the software automatically normalizes the higher level records to the level of the other data records.

Thus, the list of qualified records for GENDER EQ FEMALE is replaced with a new list of records consisting of JOB SKILLS records that are descendants of the qualified records for GENDER EQ FEMALE. Arrows point to the new records, and as before, the numbers represent Hierarchical Table addresses of the records.



Now the two lists can be compared. When the following expression is processed, only the record with address 234 qualifies:

```
SKILL TYPE EQ COBOL AND GENDER EQ FEMALE
```

When the OR operator is used, the following records qualify: 234, 345, 487, and 987.

```
SKILL TYPE EQ COBOL OR GENDER EQ FEMALE
```

If the ENTRY record with address 375 had no JOB SKILLS record as a descendant, the software would have included that record in the list of qualified records for the following expression:

```
SKILL TYPE EQ COBOL OR GENDER EQ FEMALE
```

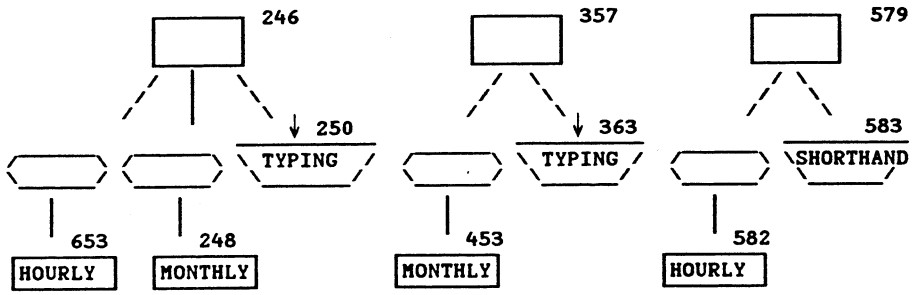
Notice that, in this case, normalization is done automatically. In other words, the process is transparent.

- **Case 4**
  Each condition specifies items from disjoint schema records. For example, suppose that you want those employees that know typing and work on an hourly basis. At first, it seems that the following expression would yield the desired results.

```
SKILL TYPE EQ TYPING AND PAY SCHEDULE EQ HOURLY
```

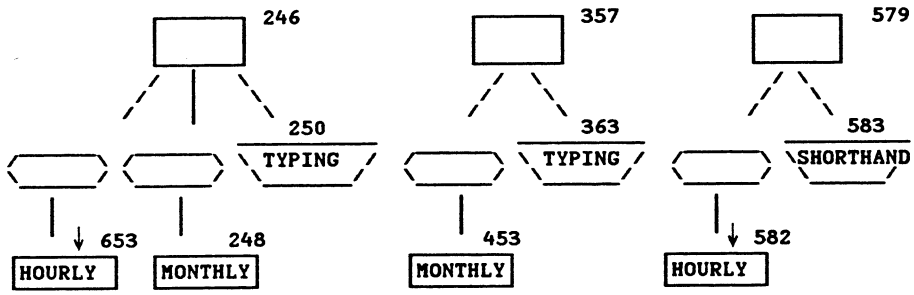
However, as shown below, this expression cannot be processed because the conditions specify items from disjoint schema records. The example includes three data trees. The numbers represent Hierarchical Table addresses, and arrows point to the data records that qualify for the following condition:

```
SKILL TYPE EQ TYPING
```



Arrows point to the data records that qualify for the condition

```
PAY SCHEDULE EQ HOURLY
```



In trying to process either of the following expressions

```
SKILL TYPE EQ TYPING AND PAY SCHEDULE EQ HOURLY

SKILL TYPE EQ TYPING OR PAY SCHEDULE EQ HOURLY
```

the software finds a problem similar to the one found in Case 2. However, because it recognizes that the records in the expressions are disjoint, the software cannot process the expression and issues an error message.

There are three acceptable ways of expressing the criteria mentioned above. The choice depends on what type of record you want in the list of qualified records for the expression.

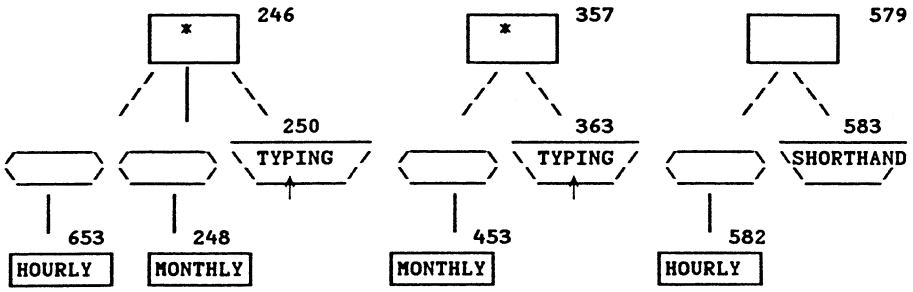
a) If the qualified records are to be similar to the SALARY WITHIN POSITION record, then you would use the following expression:

```
ENTRY HAS SKILL TYPE EQ TYPING AND PAY SCHEDULE EQ HOURLY
```

First, the software finds the records that satisfy this expression:

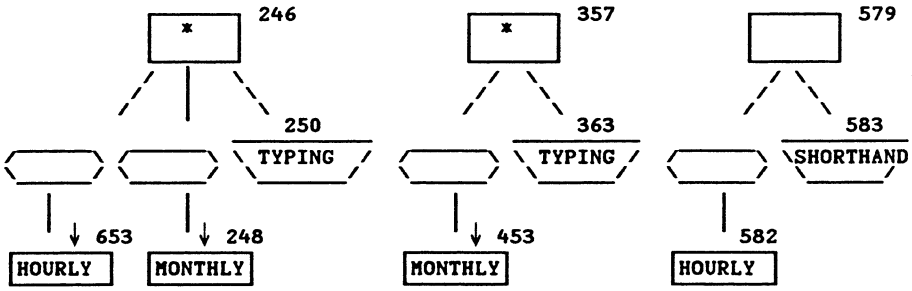
```
ENTRY HAS SKILL TYPE EQ TYPING
```

It processes the expression by finding the ENTRY records that are ancestors of the qualified records for the condition SKILL TYPE EQ TYPING. Asterisks indicate the qualified ENTRY records.



The software then finds those SALARY WITHIN POSITION records that are descendants of the ENTRY records qualified for the following expression:

```
ENTRY HAS SKILL TYPE EQ TYPING
```



At this point, the software can evaluate the expression

```
ENTRY HAS SKILL TYPE EQ TYPING AND PAY SCHEDULE EQ HOURLY
```

The SALARY WITHIN POSITION records that are descendants of the records qualifying for ENTRY HAS SKILL TYPE EQ TYPING are 248, 453, and 653. The records qualifying for PAY SCHEDULE EQ HOURLY are 582 and 653. Therefore, the SALARY WITHIN POSITION record that satisfies the above expression is 653.

The SALARY WITHIN POSITION records that satisfy the expression ENTRY HAS SKILL TYPE EQ TYPING OR PAY SCHEDULE EQ HOURLY are 248, 453, 582, and 653.

b) If you want the qualified records to be similar to the JOB SKILLS record, use the following expression:

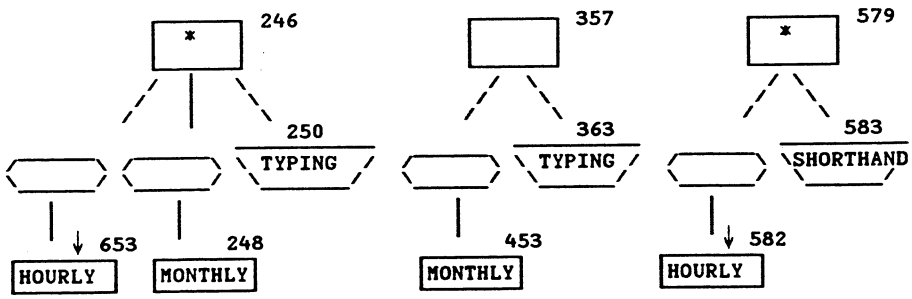
```
SKILL TYPE EQ TYPING AND ENTRY HAS PAY SCHEDULE EQ HOURLY
```

First, the software finds the records that satisfy this expression:

```
ENTRY HAS PAY SCHEDULE EQ HOURLY
```

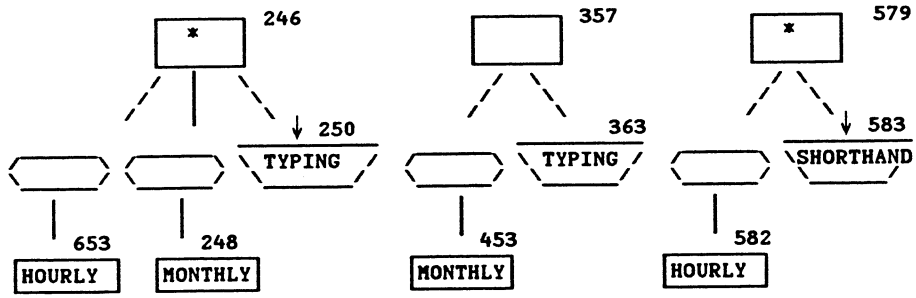
It first processes the expression by finding the ENTRY records that are ancestors of the qualified records for the following condition:

```
PAY SCHEDULE EQ HOURLY
```



The software then finds those JOB SKILLS records that are descendants of the ENTRY records qualified for the following expression:

```
ENTRY HAS PAY SCHEDULE EQ HOURLY
```



At this point, the software can evaluate the expression SKILL TYPE EQ TYPING AND ENTRY HAS PAY SCHEDULE EQ HOURLY.

The JOB SKILLS records that are descendants of the records qualifying for ENTRY HAS PAY SCHEDULE EQ HOURLY are 250 and 583.

The JOB SKILLS records that qualify for SKILL TYPE EQ TYPING are 250 and 363. Therefore, the JOB SKILLS record that satisfies the above expression is 250.

The JOB SKILLS records that satisfy the expression SKILL TYPE EQ TYPING OR ENTRY HAS PAY SCHEDULE EQ HOURLY are 250, 363, and 583.

c) If you want the qualified records to be level 0 ENTRY records, use the following expression:

```
ENTRY HAS SKILL TYPE EQ TYPING
   AND ENTRY HAS PAY SCHEDULE EQ HOURLY
```

The first example under a) shows the records that qualify for the expression ENTRY HAS SKILL TYPE EQ TYPING. The first example under b) shows the records that qualify for the expression ENTRY HAS PAY SCHEDULE EQ HOURLY.

To process the expression with the logical operator AND, the software finds the records that appear in both of the lists below:

```
ENTRY HAS SKILL TYPE EQ TYPING      246 and 357
ENTRY HAS PAY SCHEDULE EQ HOURLY    246 and 579
```

Therefore, record 246 satisfies the expression with AND.

The expression ENTRY HAS SKILL TYPE EQ TYPING OR ENTRY HAS PAY SCHEDULE EQ HOURLY is satisfied by records 246, 357, and 579.

In summary, normalization plays an important role in the processing of expressions. When normalization is done automatically, as in Case 3, it is called *implicit* normalization. When normalization occurs because the HAS operator is specified, as in Cases 2 and 4, it is called *explicit* normalization.

Notice that in the examples shown in Cases 2 and 4, the ENTRY record preceded the HAS operator. The schema record that is specified with the HAS operator plays a special role in the processing of retrieval and update commands. That record is called the *focal* record for the expression.

Chapter 26

Optimization of Where-Clauses

WHERE-CLAUSE PROCESSING 26-2
Processing of Indexed Expressions 26-2
Processing of Non-Indexed Expressions 26-3
Processing of Partially Indexed Expressions 26-4

FACTORS AFFECTING THE EFFICIENCY OF WHERE-CLAUSE PROCESSING 26-5

GUIDELINES FOR OPTIMIZING WHERE-CLAUSES 26-6

This chapter provides information about constructing where-clauses for optimal processing. The discussion applies primarily to those where-clauses that are to be executed repeatedly and should, therefore, be designed for efficient processing, for example, where-clauses that are stored in strings.

WHERE-CLAUSE PROCESSING

The following discussion focuses on the difference between where-clause processing that uses the index to find qualified data records and where-clause processing that does not use the index.

A where-clause can be either an indexed expression, a partially indexed expression, or a non-indexed expression. SYSTEM 2000 software uses the index to process an indexed expression, but not to process a non-indexed expression. It uses the index to partly process a partially indexed expression.

Processing of Indexed Expressions

After the software determines that a command is syntactically correct, it processes the where-clause. In the case of an indexed expression, it processes the conditions from right to left. To process each condition, the software uses the index to make a list of the Hierarchical Table addresses of the data records that satisfy the condition. These lists are subsequently modified by the various operators that either join (AND and OR) or modify (AT, HAS, and NOT) the conditions. The final list contains the Hierarchical Table addresses of the qualified data records for the where-clause. For example, consider the following command, which refers to Logical Entry Two in the foldout at the end of this manual:

```
PRINT C1, C105, C111 WHERE C111 GE 1500.00 AND C105 EQ 03/01/1989:
```

The Hierarchical Table address of the C100 data record that satisfies C105 EQ 03/01/1989 is 13.

The Hierarchical Table addresses of the C110 data records that satisfy C111 GE 1500.00 are 38, 14, 15, and 17.

To join the two lists, the first list is normalized down; that is, the address of the C100 record is replaced by 38, 14, and 15.

The AND operator is now applied to the two lists, that is,

(14, 15, 17, 38) AND (14, 15, 38) = (14, 15, 38).

These are the Hierarchical Table addresses of the qualified data records. The output is shown below.

```
1*  1120
  105* 03/01/1989
    111*   $1,950.00          ← from record 38
1*  1120
  105* 03/01/1989
    111*   $1,800.00          ← from record 14
1*  1120
  105* 03/01/1989
    111*   $1,650.00          ← from record 15
```

Processing of Non-Indexed Expressions

Non-indexed expressions involve a full pass of the database, and the expression is repeatedly evaluated to obtain the qualified data records.

In other words, the software goes through the database (the Hierarchical Table and the Data Table) in logical order and without considering those records that are below the level of the lowest record referred to in the expression. As each lowest record is reached, a path is established for that record and the expression is evaluated along that path. This evaluation is done by testing each condition in the expression as it occurs, that is, from left to right. If the records in the path satisfy the expression, the lowest record becomes a qualified data record for the expression.

For example, consider the following command, which refers to Logical Entry Two in the EMPLOYEE database foldout:

```
PRINT C1, C101, C114 WHERE C101 EQ PROGRAMMER AND C114 GE 150.00:
```

Both C101 and C114 are non-key items; therefore, the two conditions are non-key, and the expression is a non-indexed expression. To process the where-clause, the software "walks" through the database. (In this example, only the portion of the process for Logical Entry Two is shown.)

Starting at record 12, the software proceeds in logical order down to record 38. Since C110 is the lowest record referred to in the expression, the software does not consider C120 records. At record 38, it evaluates the expression along the path of record 38, that is, 38, 13, and 12. Since records 13 and 38 satisfy the expression, record 38 becomes a qualified data record for the expression.

The next path to be considered is the path for record 14, that is, 14, 13, and 12. Again, the expression is satisfied, and record 14 becomes another qualified data record.

Evaluation of the next path yields record 15 as a qualified data record.

The next path is the one for record 17, that is, 17, 16, and 12. In this case, the value for C101 in record 16 does not satisfy the first condition and thus no record qualifies.

As a result, the qualified data records for the expression are as follows:

14, 15, and 38

Here is the output.

```
1*  1120
  101* PROGRAMMER
    114*     $239.65
1*  1120
  101* PROGRAMMER
    114*     $215.40
1*  1120
  101* PROGRAMMER
    114*     $193.14
```

Processing of Partially Indexed Expressions

Because a partially indexed expression consists of an indexed expression and a non-indexed expression, the software combines the processes for those two types of expressions.

The first step is to scan the entire where-clause. If any key conditions are found to the left of a non-key condition, the key conditions are moved to the right. Thus, after this initial step, the where-clause consists of non-key conditions followed by key conditions. These key conditions are then processed as an indexed expression, that is, by using the index.

The result of this process is a sorted list of the Hierarchical Table addresses of data records satisfying the indexed expression. The software now considers the families of each of these records (that is, the ancestors and descendants of the records). The non-indexed portion of the where-clause is now evaluated for each path in those families. If the records in the path satisfy the non-key conditions, the lowest record becomes a qualified data record for the entire where-clause.

For example, consider the following command, which refers to Logical Entry Two in the EMPLOYEE database foldout:

```
PRINT C1, C105, C114 WHERE C114 GE 150.00 AND C105 EQ 03/01/1989:
```

Notice that C114 is a non-key item and C105 is a key item. Therefore, the expression is a partially indexed expression. In this case, the non-key condition appears to the left of the key condition so that there is no need to switch the conditions around.

To process the where-clause, the software first evaluates the key condition. In this case, the only record satisfying the condition C105 EQ 03/01/1989 is record 13. The software now considers the family of that record, that is, the set of its ancestors and descendants. The software proceeds to evaluate the non-key condition for the first path, consisting of records 38, 13, and 12. Since record 38 meets the condition, it becomes a qualified data record. The path 14, 13, and 12 is considered next, and as a result, 14 becomes a qualified data record. Finally, path 15, 13, and 12 is considered, and record 15 becomes the last qualified data record. Record 17 is not considered because it is not in the family of record 13. Thus, after sorting, the qualified data records for the where-clause are as follows:

14, 15 and 38

Here is the output.

```
1*  1120
  105* 03/01/1989
    114*     $239.65
1*  1120
  105* 03/01/1989
    114*     $215.40
1*  1120
  105* 03/01/1989
    114*     $193.14
```

FACTORS AFFECTING THE EFFICIENCY OF WHERE-CLAUSE PROCESSING

Efficiency of where-clause processing depends primarily on two factors: the placement of conditions within the where-clause and whether the conditions are key or non-key. In order to process key conditions, the software creates lists of Hierarchical Table addresses of the data records satisfying each condition. This requires scratch file space; therefore, the amount of scratch file space is a factor to be considered for where-clause optimization.

To optimize a where-clause, you need to estimate the size of the lists created for key conditions. You can use the TALLY command for this purpose. In the case of stored parametric strings, where the specified value is not known (for example, C111 EQ *1*), the average size of the list can be estimated by using the TALLY command with the ALL option.

For example, consider the following command and resulting output:

```
TALLY /ALL/ C111:

*******************************************
 ITEM-        PAY RATE
*******************************************
 MINIMUM-       $4.50
-------------------------------------------
 MAXIMUM-   $7,500.00
-------------------------------------------
       56 DISTINCT VALUES
-------------------------------------------
      142 TOTAL OCCURRENCES
-------------------------------------------
```

The average list size for C111 EQ *1* can be estimated to be about 3.

```
          2.54
     ________
56 | 142.00
```

In most cases, processing non-indexed or partially indexed expressions requires less scratch file I/O than processing an indexed expression. However, processing a non-indexed expression requires a full pass of the database in logical order. This can require extra I/O if INSERT TREE commands have caused the records to be physically scattered throughout the database.

Processing a partially indexed expression is often more efficient than processing either a non-indexed or an indexed expression. However, if the key condition in a partially indexed expression qualifies a large percentage of the records in the database, the partial pass can become almost a full pass.

When working with key conditions, the relative size of the lists generated for each condition determines whether one or more conditions should be treated as non-key (by specifying NON-KEY). This causes the expression to be processed as a partially indexed expression or a non-indexed expression. See paragraph 15 in the next section for more information.

Keep in mind that the final number of selected data records for a command is not always a good indication of how expensive it is to identify those records. More important is the amount of data processed during the intermediary steps.

GUIDELINES FOR OPTIMIZING WHERE-CLAUSES

This section presents general techniques and rules for constructing efficient where-clauses. Non-indexed expressions can be processed only when full passes are allowed. In other words, if the master password holder has specified NO FULL PASSES, then only indexed and partially indexed expressions can be processed.

1. When constructing indexed expressions, the following conditions are always treated as non-key, even when the specified item is key:

   item **FAILS**
   item * *binaryoperator* * *item*

2. When constructing partially indexed expressions, place key conditions on the right and non-key conditions on the left. Otherwise, the software has to switch the conditions around.

3. When constructing indexed expressions, consider that the software processes conditions from right to left. Therefore, it is more efficient to place conditions that generate small lists to the right. This is especially important in the case of conditions joined by the AND operator. In this case, if the software finds a condition that is not satisfied by any record, it stops processing the other conditions; that is, it knows that the intersection of an empty list with a list is also empty. For example, assume that MANY stands for a key condition satisfied by 10,000 records, SOME for a key condition satisfied by 5,000 records, and NONE a key condition that might not be satisfied by any record. The where-clause

   ```
   WHERE MANY AND SOME AND NONE
   ```

   is more efficient to process than the following where-clause:

   ```
   WHERE NONE AND SOME AND MANY
   ```

   In the first where-clause, the software would stop processing if NONE generates an empty list; lists for MANY and SOME would not be generated. However, in the second where-clause, the software would first generate lists for MANY and SOME, process MANY AND SOME, and then generate the list for NONE which, if empty, would result in NONE AND SOME AND MANY being empty.

4. The reverse situation is true for non-indexed expressions; the software processes the conditions from left to right. Therefore, it is more efficient to place conditions that generate small lists to the left.

   Just as for indexed expressions, this is especially important in the case of conditions joined by the AND operator. In this case, if the software finds a condition that is not satisfied by any record, it stops processing the other conditions. That is, when a record in the path does not meet a condition, the path is not evaluated for the other conditions.

For example, assume that MANY stands for a non-key condition satisfied by 10,000 records, SOME for a non-key condition satisfied by 5,000 records, and NONE a non-key condition that might not be satisfied by any record. The where-clause

```
WHERE NONE AND SOME AND MANY
```

is more efficient to process than the following where-clause:

```
WHERE MANY AND SOME AND NONE
```

In the first where-clause, the software would stop evaluating a path if no record in the path satisfies NONE. That is, SOME and MANY would not be evaluated for that path. In the second where-clause, a path would be evaluated for MANY and SOME, but if no record in the path satisfies NONE, the processing of MANY AND SOME would be wasted time.

5. When constructing indexed expressions, place conditions from lowest level to highest level in left-to-right order to minimize normalization.

   For example, assume that LVL-0 is a key condition specifying a level 0 item, LVL-1 is a key condition specifying a level 1 item, and so on. The where-clause

   ```
   WHERE LVL-3 AND LVL-2 AND LVL-1 AND LVL-0
   ```

   is more efficient to process than a where-clause with LVL-0 on the left and LVL-3 on the right.

6. When constructing a where-clause that includes non-key conditions, avoid specifying unnecessary parentheses, because each set of parentheses designates a separate unit to be processed.

   For example, assume that C21, C22, and C23 are non-key items and C24 is a key item. Consider the following where-clause:

   ```
   WH (C21 EQ 1 AND (C22 EQ 2 AND (C23 EQ 3 AND C24 EQ 4)))
   ```

   To process this where-clause, the software first processes C23 EQ 3 AND C24 EQ 4 as a partially indexed expression. Then the software does another partial pass of the database to evaluate C22 EQ 2. Finally, the software does a third partial pass to evaluate C22 EQ 1. Thus, the processing of the above where-clause requires three partial passes of the database. By comparison, consider the equivalent where-clause:

   ```
   WH C21 EQ 1 AND C22 EQ 2 AND C23 EQ 3 AND C24 EQ 4
   ```

   In this case, the where-clause is processed as one partially indexed expression -- all three non-key conditions are evaluated in one partial pass.

7. When possible, use a condition specifying a ternary operator instead of an equivalent expression consisting of separate conditions using binary operators. For example, the condition

   ```
   C2 SPANS ADAMS * GARCIA
   ```

   is more efficient to process than the following equivalent expression:

```
C2 GE ADAMS AND C2 LE GARCIA
```

When the AND expression is processed, two lists are created, sorted, and intersected. However, when the SPANS condition is processed, only one list is created and sorted. Furthermore, this list is usually a lot smaller than either of the lists generated for the expression.

In some cases, you can use a condition with a ternary operator instead of an expression consisting of a number of conditions joined by OR. For example, the condition

```
ZIP CODE SPANS 78701 * 78703
```

is more efficient to process than the following equivalent expression:

```
ZIP CODE EQ 78701 OR ZIP CODE EQ 78702 OR ZIP CODE EQ 78703
```

Instead of creating three lists, doing three sorts and two merges, as is the case when the previous expression is processed, the processing of the SPANS condition requires only one list and one sort. However, the list to be sorted is larger than each of the three lists. Notice that the condition is equivalent to the expression, because in this example, ZIP CODE cannot have other intermediate values besides 78702.

8. Where applicable, replace an expression using AT, AND, and EXISTS with one using AT and the HAS operator to specify downward normalization. Specifically, this rule applies to expressions similar to the one shown below, which refers to the EMPLOYEE database:

```
WH C111 EXISTS AT 1 AND C102 EXISTS AT 1 AND C2 EQ SMITH
```

To process the above expression, the software creates a list of the data records satisfying C2 EQ SMITH and intersects that list with the list created from C102 EXISTS AT 1. Generating this second list involves the processing of all C100 records containing a value for C102, in order to determine the position of each record under its parent C0 record.

Once the list for C102 EXISTS AT 1 AND C2 EQ SMITH is created (and sorted), it must be intersected with the list created for C111 EXISTS AT 1. Again, generating this list requires the processing of all C110 records containing a value for C111, in order to determine the position of each record under its parent C100 record.

So, processing the above where-clause requires the generating three lists (for C2 EQ SMITH, for C102 EXISTS, and for C111 EXISTS). The AT operator is then applied to two of them, and finally the AND operator is applied twice.

By comparison, processing the following where-clause is considerably more efficient. Notice that this where-clause qualifies C100 records even if they do not contain a value for C111 or if their parent record does not have a value for C102.

```
WH (C110 HAS (C100 HAS C2 EQ SMITH) AT 1 ) AT 1
```

In this case, the list of C0 data records satisfying C2 EQ SMITH is replaced by a list consisting of their descendant C100 records. Those C100 records that are first under their C0 parents are kept. This new list is replaced by a list consisting of their descendant C110 records. Those C110 records that are first under their C100 parents become the qualified data records for the where-clause.

9. Where possible, use the HAS operator to prevent automatic downward normalization, which takes place when AND or OR join two key-conditions at different levels. This is especially important when the target record for the action-clause corresponds to the record referred to in the higher-level condition. For example, consider the following command:

```
PR C1 WH C111 EQ 2000.00 AND C2 EQ SMITH:
```

To process the where-clause, two lists are created and sorted: the list of C0 records satisfying C2 EQ SMITH and the list of C110 records satisfying C111 EQ 2000.00. The C0 records in the first list are then replaced by their C110 descendants. This new list is sorted and intersected with the list of C110 records.

The result is a list of the qualified C110 records for the where-clause. Because C1 is specified in the action-clause, C0 is the target record. So the selected records are those C0 records that are ancestors of the C110 qualified records.

In summary, to process this command, the software normalizes the list of C0 records down to level 2 and then normalizes the list of qualified C110 records back up to level 0. In contrast, consider the following command:

```
PR C1 WH C0 HAS C111 EQ 2000.00 AND C2 EQ SMITH:
```

As before, to process the where-clause, two lists are created and sorted. The first list consists of C0 records satisfying C2 EQ SMITH and the second list consists of C110 records satisfying C111 EQ 2000.00. However, in this case, the C110 records in the second list are replaced by their C0 ancestors.

Now the two lists of C0 records are intersected to find the qualified data records for the where-clause. These C0 records become the selected records for the action-clause without any further normalization. In this case, the only normalization that takes place is the replacement of a list of C110 records by a smaller list of their C0 ancestors.

10. Before using the NOT operator, consider the following quidelines:

    a) NOT *expression* is expensive to process since it requires a full pass of the database.

    b) NOT *item* EQ *value* is equivalent to *item* NE *value* OR *item* FAILS.

    c) The following expressions are treated as indexed:

```
NOT indexed-expression AND indexed-expression
indexed-expression AND NOT indexed-expression
```

    To process this type of expression, the software removes the records satisfying the indexed expression preceded by NOT from the list of records satisfying the other indexed expression.

Note, however, that if parentheses enclose NOT *indexed-expression*, the previous expressions are treated as partially indexed expressions.

d) Although the condition *key-item* FAILS is equivalent to the expression NOT *key-item* EXISTS, keep in mind that in the context of Paragraph (c) above, when the *condition* is used, the result is a partially indexed expression, but when the *expression* is used, the result is an indexed expression. That is, the following expression is a partially indexed expression because the condition is non-key:

indexed-expression **AND** *key-item* **FAILS**

But, this expression is an indexed expression:

indexed-expression **AND NOT** *key-item* **EXISTS**

11. The following considerations apply when SAME is used.

a) SAME is not allowed in non-indexed expressions.

For example, SAME is not allowed in the where-clause below because, even though *key-condition* AND SAME is an indexed expression, the where-clause expression as a whole is non-indexed:

WH *nonindexed-expression* **OR** *key-conditions* **AND SAME**

b) The only case requiring a full pass for expressions including SAME is when a where-clause consists only of NOT SAME. In this example C8 is a key item.

```
PRINT C2, C8 WH C8 EQ CAUCASIAN:
PRINT C2, C8 WH NOT SAME:
```

A full pass of the database is required to process the where-clause in the second command.

c) Other uses of NOT SAME are not allowed, except when joined to an indexed expression with AND. In that case, as discussed in Paragraph 10 (c), the where-clause expression is indexed.

12. When specifying the HAS operator before an indexed expression, examine the conditions in the expression to determine whether it would be more efficient to treat the conditions as non-key.

For example, suppose C121 is a key item. Consider the processing steps for the following expression:

```
WH C0 HAS C121 EXISTS
```

To process this expression, the software uses the index to create a list of all C120 records that have values for C121. That list is sorted, and a new list consisting of the C0 ancestors of those C120 records is generated. Finally, this list of C0 records is sorted, and duplicate records are deleted.

Also, consider the steps involved in processing this expression as non-key:

```
WH C0 HAS NON-KEY C121 EXISTS
```

To process this expression, the software evaluates the condition along the path of the first C120 record under the first C0 record in the database. If that C120 has a value for C121, its C0 ancestor becomes a qualified data record. The software proceeds to evaluate the condition along the path of the first C120 record under the second C0 entry. No other paths are considered in the first logical entry.

If a C120 record does not have a value for C121, the software evaluates the condition along the path of the next logical C120 record. The process continues until all logical entries are examined.

13. The following considerations pertain to the AT operator:

    a) An indexed expression of the form

       key-condition **AT** *n*

       where the condition does not refer to a level 0 item becomes increasingly expensive to process as *n* increases. It is most expensive to find the last position (n = 0). For example, consider the following expression as it applies to Logical Entry Two shown in the EMPLOYEE database foldout:

       ```
       WH C111 EXISTS AT 3
       ```

       To process this expression, the software creates a list with the Hierarchical Table addresses of the records satisfying the condition C111 EXISTS. In this case, the list consists of 17, 15, 14, and 38. After sorting, the list becomes: 14, 15, 17, and 38.

       When the AT operator is applied, the software takes each record, finds its parent, and determines the third child of that record. If the address of the third child is the same as the address of the record in the list, that record becomes a qualified data record for the expression. In this case, only record 15 qualifies.

    b) When the condition in the above expression refers to an item in C0, the process is often more expensive, since for each record that satisfies the condition, the software must go through the Hierarchical Table to find the *n*th C0 record. However, in this case, AT 0 is least expensive to process because the Hierarchical Table entry for the first C0 record contains the address of the last C0 record in the database.

    c) In contrast to the extra processing involved when the AT operator is included in an indexed expression, the inclusion of the AT operator in a non-indexed expression does not require additional processing. This is because the software keeps track of the position of every data record as it passes through the database. In fact, the software checks the position first and evaluates the condition preceding AT only for those data records at the specified position. However, for non-indexed expressions, AT 0 is more expensive to process than AT *nonzero*, because the software determines that a record is the last one under its parent only after it discovers that the set of right siblings of the record contains no other similar sibling.

14. When you specify a text search condition for a key item, the software searches the index. Normally, index searches are accomplished by a binary search on a specific value, which is possible because the index is maintained in internal collating sequence. However, because text searching means looking for a string of characters within a value, a binary search cannot be used.

    Instead, each value in the index for that item is read to search for the string. The processing time varies depending on the number of different values for that item, which you can determine by issuing the TALLY command.

    In general, it is more efficient to process the text search condition as non-key and join it to a key condition using the AND operator. For example, in the case of the EMPLOYEE database, the where-clause

    ```
    WH NON-KEY C2 CONTAINS SON AND C102 EQ MARKETING
    ```

    is more efficient to process than this where-clause:

    ```
    WH C2 CONTAINS SON
    ```

15. When constructing an expression with key conditions, you can specify that one or more conditions be treated as non-key (by preceding a condition with NON-KEY). specifying NON-KEY decreases the time required to process some expressions. Each of these situations is discussed below.

    The following notation applies throughout this discussion:

    Capital letters stand for key conditions. SR-*capital letter* stands for the schema record referred to in the condition. For example, SR-A stands for the schema record referred to in condition A. s[*capital letter*] stands for the size of the list generated for that condition. For example, s[A] is the number of data records satisfying A.

    a) Two conditions joined by AND.

    Case 1 A AND B, where A and B refer to items in the same schema record. In this case, processing time is a function of the size of the list generated for each condition.

    Specify NON-KEY A AND B
    when s[A] > s[B] and s[A]/s[B]> t[1]

    Specify NON-KEY B AND A
    when s[B] > s[A] and s[B]/s[A]>t

    For example, in one installation, it was determined that t = 50. If s[A] = 14000 and s[B] = 250, the expression should be specified as NON-KEY A AND B, since 14000/250 = 56 and 56 >50. If,

[1] t is a site-specific number (or threshold) that can be determined by issuing several commands with conditions that generate lists of various sizes and by keeping a log of how long each command takes to process. The time can be determined by using the TIMING ON command discussed in Chapter 45, "Monitoring Performance: TIMING."

however, s[A] = 9000 and s[B] = 8300, then both conditions should be specified as key.

Case 2 H AND L, where H and L refer to conditions on items at different levels. L refers to the condition for the lower level record and H refers to the condition for the higher level record.

In this case, processing time is a function of the size of the lists generated for each condition and of the average number of SR-L descendants of each SR-H record that satisfies condition H. This last value can be obtained by issuing the following command:

```
PRINT AVG (COUNT SR-L BY SR-H) WH H:
```

For example, for the EMPLOYEE database, the command

```
PRINT AVG (COUNT C120 BY C100) WH C102 EQ MARKETING:
```

gives the average number of C120 children of each C100 record with a value of MARKETING for C102.

This value is denoted by *Avg L per H* in the discussion below:

Specify L AND H
  when s[H] x (Avg L per H)/s[L] < t

Specify NON-KEY H AND L
  when s[H] x (Avg L per H)/s[L] > t

Specify NON-KEY L AND H
  when (Avg level-L per H)/s[H] < t

where *Avg level-L per H* is the average number of data records that are at the same level as SR-L and are descendants of the SR-H records satisfying condition H.

For example, suppose that t = 20. If s[H] = 400, s[L] = 250, and Avg L per H = 15, the expression should be specified as NON-KEY H AND L, since

```
400 x 15 = 6000 and 6000/250 = 24 > 20.
```

b) More than two conditions joined by AND.

When you have three or more key conditions, all referring to the same schema record, they should be processed either as all key or as all non-key except for one condition.

Specify W AND X AND Y AND Z
  when s[W] > s[X] > s[Y] > s[Z] and s[W] /s[Z] < t

Specify NON-KEY Y AND NON-KEY X AND NON-KEY W AND Z
  when s[W] > s[X] > s[Y] > s[Z] and s[W] /s[Z] > t

That is, if the where-clause contains a condition whose list is considerably smaller than the others, that condition should be specified as key, the others as non-key.

c) Conditions joined by AND and OR.

Case 1 No parentheses are specified in the following where-clause:

```
A AND B OR C
```

In this case A AND B is processed first, so the only feasible optimization is to process either A or B as non-key. That is, to optimize the processing of A AND B, refer to the previous discussion on optimizing two conditions joined by AND. Condition C should not be processed as non-key because that would result in a full pass of the database.

Case 2 Here parentheses are used:

```
A AND (B OR C)
```

If either B or C is processed as non-key, the software processes the expression in parentheses as non-key. Basically, after the conditions in parentheses are processed, the above expression is similar to an expression consisting of two conditions joined by AND. The discussion for that case also applies here. The size of the list for the expression (B OR C) can be estimated by considering that if s[B] $<$ s[C] , then

$$s[C] \leq s[\ (B\ OR\ C)\ ] \leq s[B] + s[C]$$

Supplementary Capabilities

Changing the Date Format

Displaying the Data Base Definition

Padding the Distinct Values Table

Obtaining the Current Date and Time

Controlling Processing

Repeating Command Units

Obtaining Calculations

Obtaining Arithmetic Statistics

Using Stored Strings and Functions

Chapter 27

Changing the Date Format: DATE FORMAT IS

Format 27-1
Considerations 27-1
Examples 27-2

At the beginning of each session, the date format is set to MM/DD/YYYY for input and output of dates. With the DATE FORMAT IS command, you can specify alternate input and output date formats. (The internal storage of dates is not changed. See *SYSTEM 2000 DEFINE Language, Version 12, First Edition* for specific details regarding the internal storage format for dates.)

When you issue a DATE FORMAT IS command, the alternate format remains in effect for any database opened during that session until you either issue another DATE FORMAT IS command or exit the software.

An alternate date format affects the input and output date formats for only the user who issued the command. In other words, if six users are using the same database, any DATE FORMAT IS command changes the input and output date format for only the user who issued the command, without affecting any other user.

Format

```
DATE FORMAT IS  |MM/DD/YYYY  :
                |MM/DD/YY
                |DD/MM/YYYY
                |DD/MM/YY
                |YY/MM/DD
                |YYYY/MM/DD
```

Considerations

- Regardless of the specified year format (YYYY or YY), you can enter year data as four-digit or two-digit numbers. When the input form YYYY is used, any year from 1582 (the start of the Gregorian Calendar) to 2499 can be entered. When the input form YY is used, the software assumes 19 (the 20th century).

- When displaying output, if the format YY is used, the software assumes the 20th century. If the format YYYY is used, the software displays the four digits.

- Dates prior to the beginning of the Gregorian Calendar, October 15, 1582, are not accepted by the software regardless of the date format specified. See *SYSTEM 2000 DEFINE Language, Version 12, First Edition* for information about the Gregorian Calendar and its use in the software.

- The DATE FORMAT IS command also changes the format of the system date that is printed below the title in the output from a LIST command when a title specification is included in the command.

Examples

These examples show how the specified date format affects date output. The first PRINT command is issued when the default date format is in effect.

```
PRINT BIRTHDAY WHERE EMPLOYEE NUMBER EQ 1327:

5* 02/25/1964

DATE FORMAT IS MM/DD/YY:

PRINT BIRTHDAY WHERE EMPLOYEE NUMBER EQ 1327:

5* 02/25/64

DATE FORMAT IS DD/MM/YY:

PRINT BIRTHDAY WHERE EMPLOYEE NUMBER EQ 1327:

5* 25/02/64

DATE FORMAT IS DD/MM/YYYY:

PRINT BIRTHDAY WHERE EMPLOYEE NUMBER EQ 1327:

5* 25/02/1964

DATE FORMAT IS YY/MM/DD:

PRINT BIRTHDAY WHERE EMPLOYEE NUMBER EQ 1327:

5* 64/02/25
```

Chapter 28

Displaying the Database Definition: DESCRIBE

Formats 28-2
Considerations 28-6
Examples 28-6

The DESCRIBE command displays all or part of a database definition. The DESCRIBE command

- can display an entire database schema, including header information about the history of the database
- allows the master password holder or a secondary password holder to request all or part of the database definition
- can display a single component or a range of components
- can display stored string, stored function, and Collect File items
- with the DEFINE option produces DESCRIBE output in a form that is acceptable as input to the DEFINE processor
- with the AUTHORITIES option displays component access authorities for the current password
- with the RECORD option can display a specified record definition
- with the DEFINE and AUTHORITIES options (combined) can produce DEFINE output that includes component access authorities for the current password.

Formats

```
DESCRIBE:

DESCRIBE  |[/DEFINE/]                          |component [THROUGH component] :
          |[/AUTHORITIES [n]/]                 |RECORD unit
          |[/DEFINE,AUTHORITIES/]

DESCRIBE  [/DEFINE/]  |STRINGS   :
                      |FUNCTIONS

DESCRIBE /COLLECT FILE [,DEFINE]/  [cf-item [(CF)]

   [THROUGH cf-item [(CF)]]] :
```

component is an item, record, string, or function name or component number.

n is a positive integer indicating the column in which component authorities are printed. The value of *n* must be between one and the line length, minus three. The default is 68.

unit is the component name or number of a schema record or item for the RECORD option. If you specify an item, the software displays the parent schema record along with its items and subordinate records.

cf-item is a Collect File item name.

Abbreviations:

AUTHORITIES = AUTH
COLLECT FILE = CF
DESCRIBE = DESC
THROUGH = THRU

Displaying entire definition The master password holder or a secondary password holder can display the entire database schema, which includes schema records and items. (Stored strings and functions, however, are not displayed.)

```
DESCRIBE:
```

The schema definition is displayed in logical order. The component C0 (ENTRY) is displayed only if the component name, ENTRY, has been changed to another name. Items are listed within their records in the order in which they were defined. Descendant records and their member items are listed after items belonging to the parent record. Sibling records are also listed in the order in which they were defined. The components displayed are those for which you have password authority. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for information about password security.)

This DESCRIBE command also displays the following header information:

- system release number
- database name
- definition number
- database cycle number.

Displaying Collect File items Likewise, DESCRIBE/COLLECT FILE/ displays the definition of all Collect File items. You cannot mix database components and Collect File items in one DESCRIBE command.

Because Collect file items do not have explicit numbers, the software assigns consecutive numbers (starting with 1 and incrementing by one) to the Collect File items being described. Also, all Collect File items are key items and are at level 0.

Displaying stored strings and functions You can display stored strings and functions by issuing the following commands:

```
DESCRIBE STRINGS:

DESCRIBE FUNCTIONS:
```

Displaying specific components You can use the DESCRIBE command to display the definition of individual components by including the component name or number after the keyword DESCRIBE, for example,

```
DESCRIBE C1:
```

The software prints the stored definition of each requested component for which your password has authorization.

You can also use the DESCRIBE command to display a sequence of components in the database definition. The following command displays the definition for the C100 record and all its member items.

```
DESCRIBE C100 THROUGH C106:
```

Note that the first component name or number contained in the command must appear in the definition before the second component name or number listed in the command. Also, you can use this format to display the entire database definition.

In the following command, *component* is the component name or number of the last declared string.

```
DESCRIBE CO THROUGH component:
```

This command displays the entire database schema, the stored function definitions, and the stored string definitions.

Regardless of the order in which they were defined, the software first stores the schema definition, then the function definitions, and finally string definitions.

When you specify a sequence of components with one of these formats

```
DESCRIBE component1 THROUGH component2:

DESCRIBE cf-item1 THROUGH cf-item2:
```

component1 (or *cf-item1*) must appear before *component2* (or *cf-item2*) in DESCRIBE output. The sequence can range over schema components, functions, and strings for the database, or over Collect File items.

Producing DEFINE output The DEFINE option produces DESCRIBE output that can be read directly by the DEFINE processor. You can specify selected (or all) schema components, strings, and functions, or Collect File items.

If you set the Report File to an alternate file, the DEFINE output that is written to the alternate Report File can then be used as a Command File for a DEFINE session. In this way, portions or all of an existing database definition can be used to create another.

With the DEFINE option, the command terminator appears at the end of each definition. Each line of output is 80 characters with no indention. The delimiter for string and function definitions is a special character not contained in the definition of the string/function (instead of left and right parentheses).

You can also use the DEFINE option for Collect File items. The Collect File items are automatically level 0 items, numbered sequentially starting with 1. This allows you to create a new database with the Collect File items.

Displaying component access authorities The AUTHORITIES option displays the component access authorities only for schema records and schema items for the current password. The output includes the component number, the component name, the picture, and any record membership.

Three types of component access authorities exist:

- R-authority for retrieval
- U-authority for update
- W-authority for where-clause usage.

The master password holder has all authorities for each record and item in a database. The DBA password holder has none of these authorities, and secondary password holders can have any combination or none for a component.

In AUTHORITIES output, any combination of these authorities is displayed after the picture (or any record membership). The authorities appear in a column specified by you or in column 68, by default. In either case, if a component description overlaps the column, the authorities appear on the next line in the specified column or in column 68.

Each record level is indented and, if the specified column is less than the amount of the indention, the software sets the column to the indention. Because the amount of the indention varies between record levels, the authorities line up with the component name and not with the other authorities.

The authority that a password has for a component appears as a letter, displaying in the order R, U, and W. If the password does not have one of the authorities, a period appears instead of a letter.

You cannot use the AUTHORITIES option with the COLLECT FILE option or when describing strings and functions.

If *n* is greater than the line length minus three, the software displays the authorities in column 68 when you specify the AUTHORITIES option.

The AUTHORITIES option is intended for secondary password holders, but the master password holder and the DBA password holder can issue the option. For the master password, RUW is displayed for all components.

Displaying record definition The RECORD option lets you display the definition of a schema record and the items in it. The output also includes descriptions of any schema records that are part of the specified record, but not the items in those subordinate records. No header information appears.

If you specify an item, DESCRIBE RECORD displays the parent schema record along with its items and subordinate records.

If the current password is the master password, all relevant components are displayed. If the current password is a secondary password, only the components authorized for that password are displayed. Since the DBA password cannot access the definition, it cannot issue DESCRIBE RECORD.

Combining the DEFINE and AUTHORITIES options You can combine the DEFINE and AUTHORITIES options to retrieve the following information in order to define a new database:

- item, record, string, or function definitions in a format suitable for DEFINE input, along with the MAP command at the end of the definitions
- the CONTROL command and appropriate VALID PASSWORD IS and ASSIGN commands to set up the passwords and authorities for components selected in the DESCRIBE command.

You can specify the /AUTHORITIES,DEFINE/ combination in any DESCRIBE command, that is, for one or more components, all components, strings, functions, or records. If the DESCRIBE command produces no component definitions or if the database has no secondary passwords, the MAP and CONTROL commands do not appear.

You can specify /DEFINE,AUTH/ with the keywords in either order, separated by a comma.

The master password holder can access all components and all secondary passwords and their authorities. If you are using a secondary password, only the authorized components and their authorities appear.

Considerations

- If a component name consists of several words, one of them being either the word THROUGH or the word THRU (for example, THROUGH STREETS), specify the component number instead of its name in a DESCRIBE command. This consideration does not apply to component names, such as THROUGHPUT or THRUWAY, but it does apply to THROUGH-STREETS or THRU-WAY, for example.

- Only two format options affect DESCRIBE output: SINGLE SPACE and DOUBLE SPACE. See Chapter 15, "Changing Output Format: Format Options," for format option information.

- When you are using the /DEFINE/option, the ECHO/*string*/ command is useful for producing related commands, such as MAP. For more information about the ECHO command, see Chapter 43, "Echoing Commands: ECHO."

- If the DBA password holder issues the DESCRIBE command, the software displays only the header information.

Examples

The following examples use the EMPLOYEE database.

Example 1

The output from the following DESCRIBE command is shown in the foldout at the end of this manual.

```
DESCRIBE:
```

Example 2

The following DESCRIBE commands specify a single item. In the first command, the notation (CF) indicates that it is a Collect File item; it happens to be the first item in the Collect File. The second command specifies C13, a schema item.

```
DESCRIBE /CF/ SECURITY (CF):

1*  SECURITY (INTEGER NUMBER 999)

DESCRIBE /DEFINE/ C13:

13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES ):
```

Example 3

This DESCRIBE command specifies a record as the component. Notice that only the definition of the record is displayed, not its member items.

```
DESCRIBE C110:

110* SALARY WITHIN POSITION (RECORD IN 100)
```

Example 4

This DESCRIBE command specifies a series of components using THROUGH. (If the DEFINE option had been specified, each definition would end with a command terminator and indention would not occur.)

```
DESCRIBE C300 THROUGH C420:

300*  PERSONAL INTERESTS (RECORD)
  301*  INTEREST (CHAR X(12) IN 300 WITH  FEW FUTURE OCCURRENCES )
  302*  AFFILIATION (NON-KEY CHAR X(5) IN 300)
  303*  COMMENT (NON-KEY TEXT X(5) IN 300)
400*  EDUCATIONAL BACKGROUND (RECORD)
  410*  EDUCATION (RECORD IN 400)
    411*  SCHOOL (CHAR X(15) IN 410)
    412*  DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH  FEW FUTURE OCCURRENC
          ES )
    413*  DATE COMPLETED (DATE IN 410)
    414*  MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
    415*  MINOR FIELD (NON-KEY CHAR X(12) IN 410)
  420*  TRAINING (RECORD IN 400)
```

Example 5

This DESCRIBE command specifies the AUTHORITIES option and the column in which the list of authorities should begin.

```
DESCRIBE/AUTHORITIES 45/:

SYSTEM RELEASE NUMBER  xx.x
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             2
DATA BASE CYCLE NUMBER       71
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
                                          RUW
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
                                          RUW
     3*  FORENAME (NON-KEY CHAR X(20))    RUW
     4*  HIRE DATE (DATE)                 RUW
     5*  BIRTHDAY (DATE)                  RUW
     6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11))
                                          RUW
     7*  GENDER (CHAR X(6) WITH MANY FUTURE OCCURRENCES )
         .
         .
         .
```

Example 6

This DESCRIBE command specifies the AUTHORITIES option using the default column.

```
DESCRIBE/AUTHORITIES/:

SYSTEM RELEASE NUMBER  xx.x
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             2
DATA BASE CYCLE NUMBER       71
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)                    RUW
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )     RUW
     3*  FORENAME (NON-KEY CHAR X(20))                            RUW
     4*  HIRE DATE (DATE)                                         RUW
     5*  BIRTHDAY (DATE)                                          RUW
         .
         .
         .
   100*  POSITION WITHIN COMPANY (RECORD)                         RUW
    101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)               RUW
    102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
                                                                   RUW
          .
          .
          .
```

Example 7

This DESCRIBE command specifies the AUTHORITIES option and the THROUGH option.

```
DESCRIBE/AUTH 50/ C100 THROUGH C110:

100*  POSITION WITHIN COMPANY (RECORD)          RUW
  101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
                                                RUW
  102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
                                                RUW
  103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
                                                RUW
  104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCES )
                                                RUW
  105*  START DATE (DATE IN 100)                RUW
  106*  END DATE (NON-KEY DATE IN 100)          RUW
  110*  SALARY WITHIN POSITION (RECORD IN 100)
                                                RUW
```

Example 8

This DESCRIBE command specifies the AUTHORITIES option and a column that is less than the amount of the indention for the record or item.

```
DESCRIBE/AUTHORITIES 8/:

SYSTEM RELEASE NUMBER  xx.x
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             2
DATA BASE CYCLE NUMBER       71
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
         RUW
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
         RUW
     3*  FORENAME (NON-KEY CHAR X(20))
         RUW
         .
         .
         .
   100*  POSITION WITHIN COMPANY (RECORD)
         RUW
     101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
           RUW
           .
           .
           .
       111*  PAY RATE (MONEY $9999.99 IN 110)
             RUW
             .
             .
             .
```

Example 9

This example specifies the RECORD option, requesting a description of schema record C100. The master password is the current password, so all items in the record are displayed.

```
DESCRIBE RECORD C100:

100*  POSITION WITHIN COMPANY (RECORD)
  101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
  102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
  103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
  104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCE
        S )
  105*  START DATE (DATE IN 100)
  106*  END DATE (NON-KEY DATE IN 100)
  110*  SALARY WITHIN POSITION (RECORD IN 100)
  130*  ADDITIONAL INFORMATION (RECORD IN 100)
```

Example 10

This example describes schema record C100 and all its items and displays the authorities starting in column 41. The current password is the master password.

```
DESCRIBE /AUTH 41/ RECORD C100:

100*  POSITION WITHIN COMPANY (RECORD)
                                    RUW
  101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
                                    RUW
  102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
                                    RUW
  103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
                                    RUW
  104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCE
        S )                         RUW
  105*  START DATE (DATE IN 100)     RUW
  106*  END DATE (NON-KEY DATE IN 100)
                                    RUW
  110*  SALARY WITHIN POSITION (RECORD IN 100)
                                    RUW
  130*  ADDITIONAL INFORMATION (RECORD IN 100)
                                      RUW
```

Example 11

This example shows specifying an item for the RECORD option. The master password holder is making the request, so all components are displayed. Item C102 is in record C100, so record C100 and its items appear. Notice also that the records C110 and C130 appear because they are in record C100, but items in C110 and C130 do not appear.

```
DESCRIBE RECORD C102:

100*  POSITION WITHIN COMPANY (RECORD)
  101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
  102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
  103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
  104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCE
        S )
  105*  START DATE (DATE IN 100)
  106*  END DATE (NON-KEY DATE IN 100)
  110*  SALARY WITHIN POSITION (RECORD IN 100)
  130*  ADDITIONAL INFORMATION (RECORD IN 100)
```

Example 12

This example shows the DEFINE option, which produces output that is acceptable as DEFINE output; that is, each definition ends with a delimiter. C12 is an item at level 0, so other level 0 items appear, as well as schema records C100, C200, C300, and C400 at level 1.

```
DESCRIBE /DEFINE/ RECORD C12:

   0*  ENTRY :
   1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999) :
   2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES ) :
   3*  FORENAME (NON-KEY CHAR X(20)) :
   4*  HIRE DATE (DATE) :
   5*  BIRTHDAY (DATE) :
   6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11)) :
   7*  GENDER (CHAR X(6) WITH MANY FUTURE OCCURRENCES ) :
   8*  ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES ) :
   9*  EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES ) :
  10*  OFFICE-EXTENSION (NON-KEY CHAR X(9)) :
  11*  ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99) :
  12*  ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99) :
  13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES ) :
  14*  STREET ADDRESS (NON-KEY CHAR X(20)) :
  15*  CITY-STATE (NON-KEY CHAR X(15)) :
  16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES ) :
 100*  POSITION WITHIN COMPANY (RECORD) :
 200*  JOB SKILLS (RECORD) :
 300*  PERSONAL INTERESTS (RECORD) :
 400*  EDUCATIONAL BACKGROUND (RECORD) :
```

Example 13

This example uses the secondary password PSWD, which has authority for items C1, C2, C3, C14, C15, and C16 but no authority for the C0 schema record. A secondary password can give an authorized item in a DESCRIBE RECORD command even though the password has no authority for the record.

```
DESCRIBE RECORD C1:

  1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
  2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
  3*  FORENAME (NON-KEY CHAR X(20))
 14*  STREET ADDRESS (NON-KEY CHAR X(20))
 15*  CITY-STATE (NON-KEY CHAR X(15))
 16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES )
```

Example 14

This example specifies a range of components and combines the DEFINE and AUTHORITIES options. The current password is the master password. The output consists of sixteen item definitions, the MAP and CONTROL commands, and commands for assigning the secondary password and its authorities.

```
DESCRIBE /AUTH, DEFINE/ C1 THROUGH C16:

   1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999) :
   2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES ) :
   3*  FORENAME (NON-KEY CHAR X(20)) :
   4*  HIRE DATE (DATE) :
   5*  BIRTHDAY (DATE) :
   6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11)) :
   7*  GENDER (CHAR X(6) WITH MANY FUTURE OCCURRENCES ) :
   8*  ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES ) :
   9*  EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES ) :
  10*  OFFICE-EXTENSION (NON-KEY CHAR X(9)) :
  11*  ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99) :
  12*  ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99) :
  13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES ) :
  14*  STREET ADDRESS (NON-KEY CHAR X(20)) :
  15*  CITY-STATE (NON-KEY CHAR X(15)) :
  16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES ) :
MAP: CONTROL:
VALID PASSWORD IS PSWD:
ASSIGN R,W TO C1 FOR PSWD:
ASSIGN R,W TO C2 FOR PSWD:
ASSIGN R,W TO C3 FOR PSWD:
ASSIGN R,W TO C14 FOR PSWD:
ASSIGN R,W TO C15 FOR PSWD:
ASSIGN R,W TO C16 FOR PSWD:
```

Example 15

Suppose the EMPLOYEE database contains two strings named PAYROLL INSERT and ADVERTISEMENT, and the secondary password PSWD has no authority for the strings. In this example, the master password is the current password. The DESCRIBE command produces the two string definitions, the MAP and CONTROL commands, and a command to set up the password PSWD. (No ASSIGN command appears because the secondary password does not have authority for the strings.)

```
DESCRIBE /DEFINE, AUTH/ STRINGS:

1000*  PAYROLL INSERT (STRING¢INSERT TREE C120 *1*1*1 EQ 121* *1** 122* *2**
 123* *3** 124* *4** 125* *5** 126* *6** END* WHERE C1 EQ *7*:¢) :
1001*  ADVERTISEMENT (STRING¢INSERT TREE C130 *0 EQ 131**3**132**4**END*
          WHERE C1 FAILS AND C101 EQ *1*  AND C102 EQ *2*:¢) :
MAP: CONTROL:
VALID PASSWORD IS PSWD:
```

Example 16

In this example, secondary password PSWD is the current password. The item specified in the DESCRIBE RECORD command (C1) is at level 0, and the output contains all level 0 items that are authorized to password PSWD. Commands for assigning password PSWD and its authorities appear after the MAP and CONTROL commands.

Notice that specifying a column position for display (40 in this example) causes message -365- to appear, since the software does not display the authorities.

```
DESCRIBE /DEFINE, AUTHORITIES 40/ RECORD C1:

 -365- COLUMN SPECIFICATION IGNORED WITH DEFINE OPTION -
   1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999) :
   2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES ) :
   3*  FORENAME (NON-KEY CHAR X(20)) :
  14*  STREET ADDRESS (NON-KEY CHAR X(20)) :
  15*  CITY-STATE (NON-KEY CHAR X(15)) :
  16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES ) :
MAP: CONTROL:
VALID PASSWORD IS PSWD:
ASSIGN R,W TO C1 FOR PSWD:
ASSIGN R,W TO C2 FOR PSWD:
ASSIGN R,W TO C3 FOR PSWD:
ASSIGN R,W TO C14 FOR PSWD:
ASSIGN R,W TO C15 FOR PSWD:
ASSIGN R,W TO C16 FOR PSWD:
```

Example 17

In this example, the master password holder invalidates password PSWD, so the EMPLOYEE database no longer has any secondary passwords. Message -520- appears, informing you that no passwords exist. Notice that the results of this DESCRIBE command do not include the MAP and CONTROL commands.

```
USER, DEMO: DBN IS EMPLOYEE:
CONTROL:
INVALID PASSWORD IS PSWD:
QUEST:
DESCRIBE /AUTH, DEFINE/ C1 THRU C16:

  1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999) :
  2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES ) :
  3*  FORENAME (NON-KEY CHAR X(20)) :
  4*  HIRE DATE (DATE) :
  5*  BIRTHDAY (DATE) :
  6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11)) :
  7*  GENDER (CHAR X(6) WITH MANY FUTURE OCCURRENCES ) :
  8*  ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES ) :
  9*  EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES ) :
 10*  OFFICE-EXTENSION (NON-KEY CHAR X(9)) :
 11*  ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99) :
 12*  ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99) :
 13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES ) :
 14*  STREET ADDRESS (NON-KEY CHAR X(20)) :
 15*  CITY-STATE (NON-KEY CHAR X(15)) :
 16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES ) :
-520- NO PASSWORDS -
```

Chapter 29

Padding the Distinct Values Table: ENABLE/DISABLE VALUES PADDING

Format 29-1
Considerations 29-6

To pad the Distinct Values Table, you use the ENABLE/DISABLE VALUES PADDING command. This command sets or resets a flag that is stored in the database. The software uses this flag to determine the placing of new values of key items in the Distinct Values Table. You can issue these commands only if you are the master password holder or the DBA password holder.

The default is DISABLE.

- The flag is turned off with the DISABLE option, which causes the last page of each key item index to be completely filled when an insertion is made at the end of the index. After the last page is filled, a new page is created at the end of the index.
- The flag is turned on with the ENABLE option, which causes the last page of each key item index to be partially filled when an insertion is made at the end of the index. A new last page is created to leave space for future additions in each page.

Format

```
ENABLE VALUES PADDING:

DISABLE VALUES PADDING:
```

The Distinct Values Table contains an index for each key item. This index contains an entry for each distinct value of that key item, and the values are sorted in ascending order. When a value in the index occurs in only one record in the database, the Distinct Values Table entry for that value contains a pointer to the Hierarchical Table address of the data record that the value belongs to.

For example, in the EMPLOYEE database the index for the item EMPLOYEE NUMBER would have the following entries, among others, in the order shown below:

Value	Address
1043	1
1120	12
1265	24

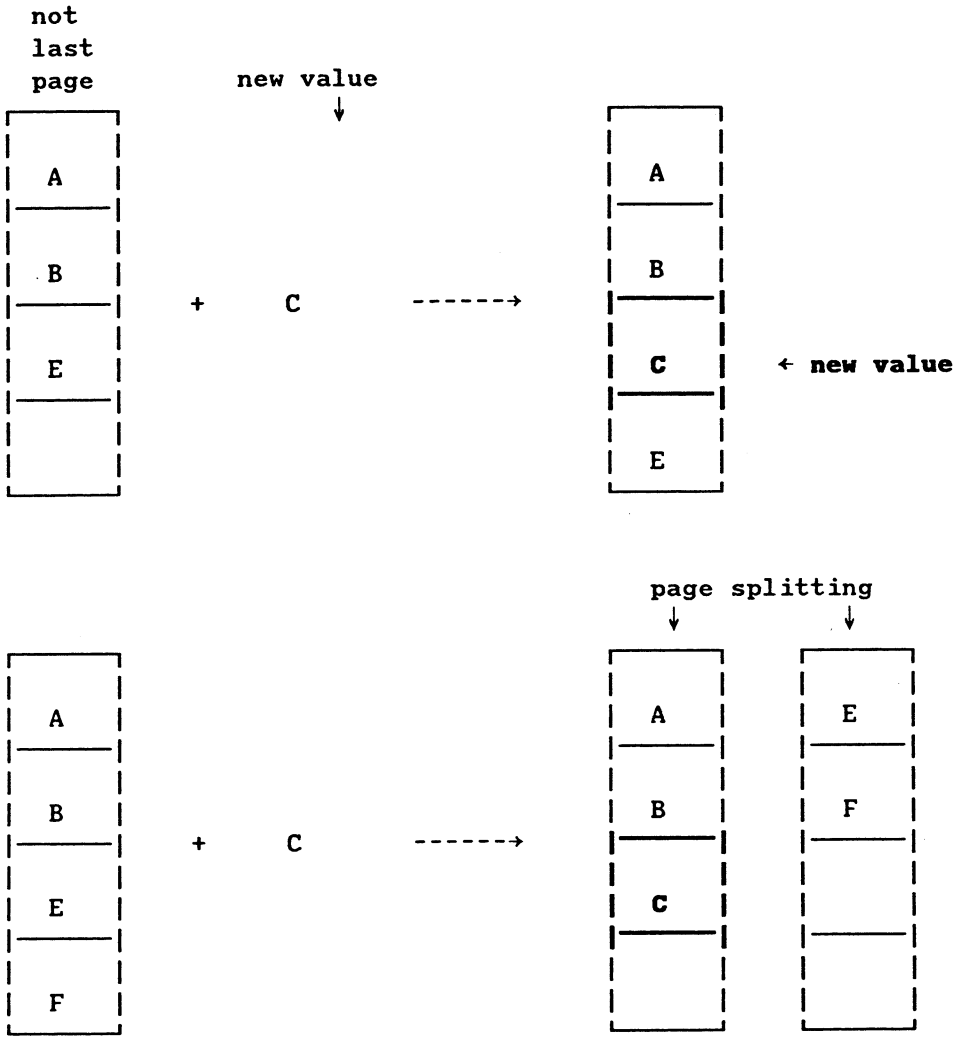
When a value occurs in more than one data record (for example, the value MALE for the key item GENDER), the corresponding Hierarchical Table addresses are stored in the Multiple Occurrence Table. The Distinct Values Table entry for the value contains a pointer to the entry in the Multiple Occurrence Table that contains the Hierarchical Table addresses.

Distinct Values Table Index for GENDER	Multiple Occurrence Table Entry Contains
MALE	12, 24
FEMALE	1

The index for an item contains one or more pages, and it grows as new values are added. Indexes of different items are always kept separate; that is, the values of different items are never mixed in a page.

When a value is to be added to a page that is not full, the higher existing values are moved down on the page, and the new value is entered after the next lower value. If the page is full, it is split into two partially filled pages, and the new value is added to the appropriate partially filled page. This is shown in Illus. 29.1 on page 29-3.

Illus. 29.1 Adding a Value in the Distinct Values Table

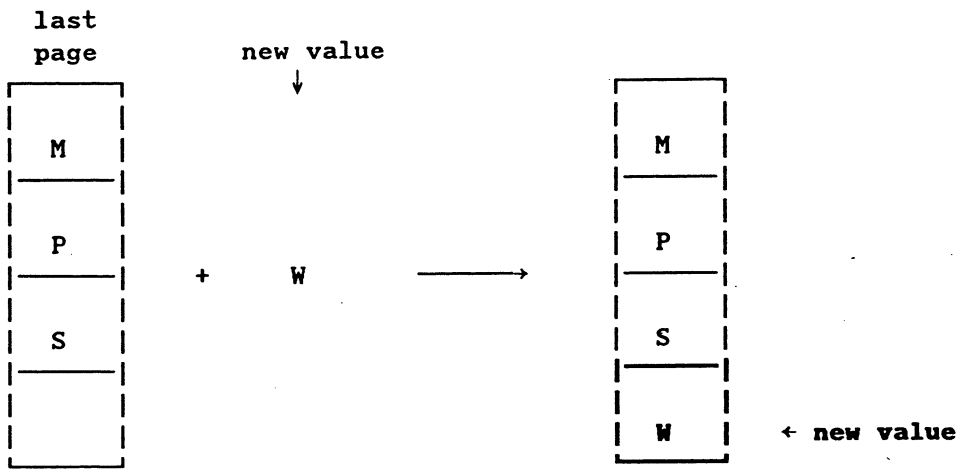


This method of inserting a new value in an index page is always true (regardless of the VALUES PADDING option chosen), except for the last page in the index for an item.

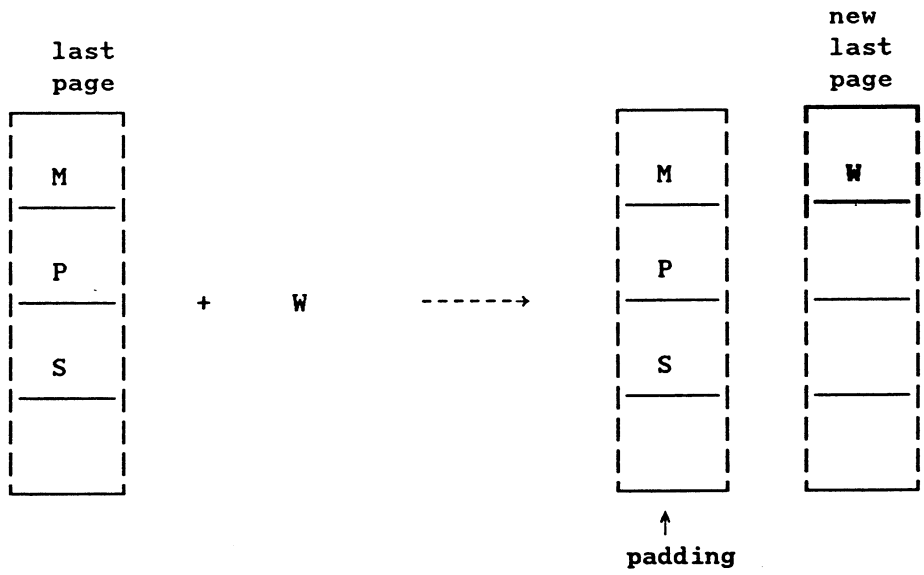
When the new value is greater than all existing values for the key item, the new value must be added at the end of the last page for the index. In this particular case, the flag set by the VALUES PADDING command determines where the value is to be inserted. If the flag is off (DISABLE), the value is added to the existing last page if there is room, otherwise to a newly created page. However, if the flag is on (ENABLE), the value is added to the existing last page only if "enough padding" will be left after adding the new value. Otherwise it is placed in a newly created page, which becomes the last page. This is shown in Illus. 29.2 on page 29-4.

Illus. 29.2 Distinct Values Table Padding

A. Flag DISABLE



B. Flag ENABLE



In other words, the Distinct Values Table padding is honored only when a new value is added at the end of the last page for a given key item. This means that if you enable distinct values padding in a job performing an initial load or a reload, each page contains the appropriate implied distinct values padding for key item values.

What is appropriate padding? The software calculates this padding as an inverse function of the padding specified for a Multiple Occurrence Table page for that item when it is defined (using the DEFINE language). Notice in the foldout that some items are defined WITH **FEW** FUTURE OCCURRENCES, WITH **SOME** FUTURE OCCURRENCES, or WITH **MANY** FUTURE OCCURRENCES. The others default to WITH **NO** FUTURE OCCURRENCES. These definitions determine the padding in the Multiple Occurrence Table, as indicated below:

Definition	Multiple Occurrence Table Padding
NO	0%
FEW	100%
SOME	400%
MANY	1400%

The software assumes that if an item has many occurrences of the same value, the item has few distinct values. Therefore, the software assigns padding for the Distinct Values Table page for that item as indicated below:

Definition	Distinct Values Table Padding	
NO	25%	(3/4 filled)
FEW	12-1/2%	(7/8 filled)
SOME	6-1/4%	(15/16 filled)
MANY	0	(full)

Thus, the ENABLE VALUES PADDING command sets the flag for the database. Once this flag is set, the last page of the index for each key item of that database includes padding of an amount equal to or greater than the amount shown above. The flag can be turned off to disallow padding by the DISABLE VALUES PADDING command.

The use of values padding offers an advantage when a large number of updates are applied to a database, so that new index pages are created by the new key values being added. As previously stated, new pages are added only when the new value is added at the end of the last page for a particular index.

Any new values not added at the end of the last index page are inserted where necessary. If the page where the new value must be inserted is full, it is split into two partially-filled pages, and the new value is added to the appropriate page. However, if new pages are being added with values padding enabled, each newly created index page has a padding factor, as indicated in Illus. 29.3 on page 29-6.

Illus. 29.3 Use of Distinct Values Table Padding

Operation	Description
LOAD	Self-Contained Facility, initial load of a database.
RELOAD	Self-Contained Facility, reload of a populated database.
CREATE INDEX	Self-Contained Facility, changing a non-key item to key.
PLEX load mode	PLEX, a load mode (LOAD/TERMINATE) session that either adds values to a new database or adds new values to the end of existing values, that is, values that are higher in ascending order than existing values for a key item.
DEFINE restructure	Self-Contained Facility, DEFINE changes to a populated database that result in a restructure of that database.

If values padding is not enabled, each new index page is full, and when a new value is added to the full pages, the overhead of page-splitting occurs.

The cases where values padding gains an advantage for later updates are indicated in the illustration above. Values padding can be enabled or disabled as the need arises for optimal performance.

Note that values padding results in more Distinct Values Table pages depending on the Multiple Occurrence Table padding specified. However, when updates are made to the database, typical turnaround time is improved because of eliminating page-splitting and fragmentation of index pages that accompanies page-splitting. Also, for key items defined with MANY FUTURE OCCURRENCES, values padding has no effect (0% Distinct Values padding), and all new index pages are full.

Considerations

- Depending on the nature of data, the use of the VALUES PADDING command can increase loading time, updating time, or storage space. However, this is extremely dependent on the database schema and the data values and whether you have performed incremental loads since the initial load.

- The Distinct Values Table contains a multi-level index. Only the lower level pages are padded. (Padding of upper level pages could prematurely force the existence of higher level pages.)

- Enabling values padding during a reload may require more space for the Distinct Values Table.

Chapter 30

Obtaining the Current Date and Time: System Strings

**NOW* 30-1*
**TODAY* 30-1*
**FTODAY* 30-2*
Format 30-2
Considerations 30-2
Examples 30-2

The date and time system strings give you the current date in date format (*TODAY*), the current date in function format (*FTODAY*), and the current time (*NOW*), according to your computer's internal calendar and clock.

NOW

NOW gives the current time in hours (00 to 23), minutes (00 to 59), and seconds (00 to 59).

You can use *NOW* in update and retrieval commands, and you can use it to time-stamp reports. You can also use *NOW* in a title specification of a LIST command to time stamp reports. However, *NOW* is treated as a CHARACTER item, so it cannot be used in functions.

The default format is HH.MM.SS.

TODAY

TODAY gives you the current date. You can use the *TODAY* system string in update and retrieval commands. The slashes in the *TODAY* format are read as division operators in arithmetical expressions, yielding a meaningless decimal number.

You cannot use *TODAY* in title specifications of a LIST command because the slashes in this case are treated as delimiters.

The default format for *TODAY* is MM/DD/YYYY.

FTODAY

FTODAY also provides you with the current date, but it substitutes periods for the slashes so that the date can appear in arithmetic expressions and in title specifications. (Slashes would be read as division operators in arithmetic expressions and as delimiters in title specifications.) You can use the *FTODAY* system string in title specifications of a LIST command, although the D-specification is more efficient for placing the current date under the title. You can also use the *FTODAY* system string in ad hoc and stored functions.

In computations, *FTODAY* is treated by the software as the number of days that have elapsed since the beginning of the Gregorian calendar on October 15, 1582.

The default format for *FTODAY* is MM.DD.YYYY.

Format

NOW	*(current time)*
TODAY	*(current date)*
FTODAY	*(current date in function format)*

Considerations

- You can change the date format with the system-wide command DATE FORMAT IS.
- You can use system strings in a Command File for updates, but they cannot be used in the Data File (for example, in the loader stream for the LOAD command).

Examples

The first two examples in this section use the EMPLOYEE database. The last two examples update a hypothetical database. For these update examples, the selected entry is shown both before and after the update command has been executed.

Example 1

This example uses *FTODAY* in an ad hoc function for the EMPLOYEE database. The ad hoc function (*FTODAY*-C4) yields the number of days each of the selected employees has worked for the company.

```
PR C1, (*FTODAY*-C4) WH C1 LT 1004:

1*  1001
10001*  1043
1*  1003
10001*  1043
1*  1002
10001*  1043
```

Example 2

This example uses *NOW* in a LIST command title specification to time-stamp a report.

```
LI/TITLE D(20)*NOW*/C1, C2, C3 WH C1 LE 1010:

                  12.25.34
                 11/09/1990

* EMPLOYEE NUMBER   LAST NAME    FORENAME
***
*            1001   WATERHOUSE   CLIFTON P.
*            1003   SALAZAR      YOLANDA
*            1002   BOWMAN       HUGH E.
*            1006   GARRETT      OLAN M.
*            1004   KNIGHT       ALTHEA
*            1007   BROWN        VIRGINA P.
*            1009   JONES        MICHAEL Y.
*            1008   HERNANDEZ    JESSE L.
*            1010   SMITH        JANET F.
*            1005   KNAPP        PATRICE R.
```

Example 3

This example uses *TODAY* in the Command File for updates. The database used is one designed to demonstrate this update command.

The first PRINT command shows the entry before the update command is issued. The second PRINT command shows the entry after the update command has been executed. Note that in the CHANGE command, asterisks must be used after the item number, to enclose the system string, and after the value being changed.

```
PR C0 WH C1 EQ A:

1* A
2*   1

  11* AX1
  12*   2

  21* AY1
  22*   3

CH C10 EQ 11**TODAY**12*2*END* WH C1 EQ A:

-342-  1 SELECTED RECORD(S) -

PR C0 WH SAME:

1* A
2*   1

  11* 11/09/90
  12*   2

  21* AY1
  22*   3
```

Example 4

This example (using the same database as Example 3, but a different entry) shows the use of *NOW* in updating a single item. The first PRINT command displays the entry prior to the update command. The second PRINT command displays the entry after the update has been executed.

```
PR C0 WH C1 EQ B:

1* B
2*   4

  11* BX1
  12*   5

  21* BY1
  22*   7

  11* BX2
  12*   6

CH C21 EQ *NOW** WH C1 EQ B:

-342-        1 SELECTED RECORD(S) -

PR C0 WH SAME:

1* B
2*   4

  11* BX1
  12*   5

  21* 18.21.47
  22*   7

  11* BX2
  12*   6
```

Chapter 31

Controlling Processing

INTRODUCTION 31-2

CONTROLLING OUTPUT VOLUME: LIMIT/END LIMIT 31-2
- *Formats 31-3*
- *Considerations 31-5*
- *Examples 31-6*

CONTROLLING EXECUTION IN EVENT OF ERRORS: STOP/CONTINUE 31-15
- *Formats 31-16*
- *Considerations 31-16*
- *Examples 31-17*

PREVENTING EXECUTION OF COMMANDS: ENABLE/DISABLE EXECUTION 31-18
- *Formats 31-18*
- *Considerations 31-18*

MOVING UPDATED PAGES TO DISK: CLEAR 31-19
  - *Damage flags 31-19*
  - *Effects of CLEAR commands 31-19*
- *Formats 31-21*
- *Considerations 31-21*
- *Example 31-23*

INTRODUCTION

Several QUEST commands allow you to control the processing of retrieval and update commands.

- The **LIMIT** command allows you to set limits on the number of records that are to be affected by retrieval or update commands.
- The **STOP** commands allow you to specify an upper limit of errors in a session. The **CONTINUE** commands countermand STOP commands within a session.
- The **ENABLE EXECUTION** command tells the software to scan retrieval and update commands so that only syntactically correct commands are executed. The **DISABLE EXECUTION** command prevents the execution of all update and retrieval commands, although it does allow the software to check for syntax errors.
- The **CLEAR/CLEAR AUTOMATICALLY** commands allow you to control when memory-resident pages modified by update operations are to be written to disk.

CONTROLLING OUTPUT VOLUME: LIMIT/END LIMIT

The LIMIT command sets a specific number of records or a minimum and maximum for the number of records on which retrieval or update commands are to be executed. Optionally, you can also control disposition (output) of selected records in retrieval commands.

- You can specify a minimum and a maximum number of records that must be selected before retrieval or update commands are executed.
- You can designate a specific number of records that must be selected before retrieval or update commands are executed; that is, the minimum limit equals the maximum limit.
- The limit can be set so there is no limit.
- You can specify whether a retrieval or update command is to be truncated or canceled if the boundaries set by the LIMIT command are not met.
- You can specify which file excess selected records will be routed to in a retrieval command.
- Limits are applied to each separate list of selected records produced by a where-clause that contains any Collect File conditions.

Each LIMIT command contains a minimum value, a maximum value, and an intermediate value. Associated with each maximum limit is a disposition parameter, and associated with each intermediate limit is a *fileref* parameter.

Formats

---

(1) LIMIT *k* :

---

minimum	=	*k*
maximum	=	*k*
intermediate	=	*k*
disposition	=	default (default = CANCEL)
fileref	=	default (default = Report File)

---

(2) LIMIT *k* , *m* [[/CANCEL] :
[[/TRUNCATE]

---

minimum	=	*k*
maximum	=	*m*
intermediate	=	*m*
disposition	=	CANCEL, TRUNCATE, or default (default = CANCEL)
fileref	=	default (default = Report File)
restriction:		*k* must be equal to or less than *m* except when *m* equals 0

---

(3) LIMIT *k* , *m* [[/CANCEL] , *j* [/*fileref*] :
[[/TRUNCATE]

---

minimum	=	*k*
maximum	=	*m*
intermediate	=	*j*
disposition	=	CANCEL, TRUNCATE, or default (default = CANCEL)
fileref	=	Valid file name or default (default = Report File)
restriction:		*k* must be equal to or less than *m* except when *m* equals 0

---

(4) END LIMIT:

---

minimum	=	0 (no limit)
maximum	=	0 (no limit)
intermediate	=	0 (no limit)
disposition	=	default (default = CANCEL)
fileref	=	default (default = Report File)

k, m, j are positive integers or 0. Zero means infinite for *m* and *j*. Also, *k* and *j* must be less than or equal to *m* except when *m* equals 0. The *j* intermediate limit has no effect on Collect File commands.

CANCEL controls the output of selected records. The default is CANCEL.
TRUNCATE That is, if the number of selected records falls outside the range specified, the command is cancelled.

fileref is a DDname declared in the JCL. The default is the Report File.

Executing a where-clause produces a set of selected records. After this set is derived, but before the action is executed, the number of selected records is compared with the limits in effect at that time.

If no LIMIT command has been issued, the default limits of zero (unlimited) are in effect. If one (or more) LIMIT command has been issued, the number of selected records is compared with the limits specified in the last LIMIT command according to the following series of tests:

1. If the number of selected records *n3* is less than "minimum," the rest of the command execution is bypassed, and the following messages are issued, where *n3 = n1 - n2*.

```
-340- n1 TOTAL SELECTED RECORD(S) -
-343- n2 RECORD(S) CANCELLED OR TRUNCATED -
-342- n3 SELECTED RECORD(S) -
```

2. If *n3* is greater than "maximum," the software issues messages -340-, -343-, and -342-, and the results of the command differ depending upon the "disposition," specified or omitted.

   a) If "disposition" equals CANCEL or is not specified (that is, it defaults to CANCEL), the command is cancelled.

   b) If "disposition" equals TRUNCATE, the software compares the list of selected records with the maximum *m*, executes the command on the first *m* selected records, and ignores the remainder. In the case of a command with an ordering-clause, truncation does not occur until after the ordering phase. Therefore, you have some control over which subset of selected records will be processed further.

3. The fileref parameter determines which file output will be routed to, depending on the relationship of the values of *intermediate*, *maximum*, and *n*.

   a) If intermediate is greater than or equal to maximum ($j \geq m$), the intermediate value and its associated fileref parameter, if given, are ignored. That is, the results are as described above.

   b) If intermediate is less than maximum ($j < m$) and *n3* is less than or equal to intermediate ($n3 \geq j$), printed output is routed to the Report File, regardless of whether an alternate fileref has been specified.

   c) If intermediate is less than maximum ($j < m$) and *n3* is greater than intermediate ($n3 > j$), printed output is routed redundantly to two files: fileref

and the Message File. The first *j* records go to the Message File, and the first *m* records go to the specified fileref or, if no fileref is specified, to the Report File.

After you issue a LIMIT command, the specifications it sets remain in effect for all subsequent commands until one of the following occurs:

- You issue another LIMIT command that changes the specifications.
- You issue an END LIMIT command.
- You exit the software.

Changing processors does not affect the LIMIT specifications in effect.

All limits are initialized to zero; that is, they are set to "unlimited" at the beginning of a session. If a maximum value is included in a LIMIT command with no disposition specified, the default is CANCEL. If an intermediate value is included in a LIMIT command and no fileref is indicated, the default is the Report File.

Considerations

- The Report File is the same as the Message File unless one or both have been renamed with a REPORT FILE IS or MESSAGE FILE IS command.
- Format (3) is not applicable for update commands. If it is in effect when an update command is executed, *j* and *fileref* are ignored, and the limits are treated as Format (2).
- Any limits in effect from the QUEST processor are honored by a MAP command containing a where-clause.
- Any limits in effect are honored by a RELOAD command containing a where-clause, regardless of whether the RELOAD command is issued in the QUEST or CONTROL language.
- If the intermediate limit is set, any LIST or PRINT command containing a where-clause with Collect File items is not allowed. An error message is issued if the where-clause contains a Collect File item.
- If limits are set, any update command with a where-clause that has Collect File items is not allowed. An error message is issued if the where-clause contains a Collect File item.
- The LIMIT command has a special relationship to the TALLY command. See Chapter 18, "Retrieval Examples by Task" for an example using the LIMIT and TALLY commands.

- If a command contains a where-clause with a Collect File condition and LIMIT is set, the limits are applied to each of the *m* Collect File records independently during the execution of the where-clause. If the limits are exceeded, where either CANCEL or TRUNCATE or both take effect for at least one of the *m* Collect File records, the following messages are issued:

```
-281- m1 TRUNCATED COLLECT FILE LIST(S) -
-282- m2 CANCELLED COLLECT FILE LIST(S) -
-340- n1 TOTAL SELECTED RECORD(S) -
-343- n2 RECORD(S) CANCELLED OR TRUNCATED -
-342- n3 SELECTED RECORD(S) -
```

  Message -281- gives the total number of where-clause lists that have been truncated, while message -282- gives the total number of where-clause lists that have been cancelled. Message -281- and -282- are optional; one or the other can appear, but not both.

  Message -340- gives the total number of records selected from all *m* Collect File records before the limits are applied. (Note that this replaces the previous message -340-, which stated TOO FEW SELECTED RECORDS or TOO MANY SELECTED RECORDS.) Message -343- gives the total number of records either cancelled or truncated from the *m1* + *m2* Collect File lists.

  Message -342- gives the final count of selected records resulting from the execution of the where-clause. Note that *n3* = *n1* - *n2*.

Examples

The following examples show the effect of different LIMIT commands on the results of like commands. The only differences in the commands involve the number of selected records that are produced as a result of various qualifications stated in the where-clauses. In addition, other forms of the LIMIT command that will produce the same results are given where applicable.

Example 1

LIMIT 3:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS- -343- 2 RECORD(S) CANCELLED OR TRUNCATED- -342- 0 SELECTED RECORD(S)-		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 1* 1002 1* 1003	
PR C1, OB C1 WH C1 LE 1005:	5	-340- 5 TOTAL SELECTED RECORDS- -343- 5 RECORD(S) CANCELLED OR TRUNCATED- -342- 0 SELECTED RECORD(S)-		

Other LIMIT commands that would achieve the same results are

```
LIMIT 3,3:
LIMIT 3,3/CANCEL:
LIMIT 3, 3, 3:
LIMIT 3, 3/CANCEL, 3:
LIMIT 3, 3, 3/ALTFILE:
LIMIT 3, 3/CANCEL, 3/ALTFILE:
```

Note: ALTFILE is the DDname.

Example 2

LIMIT 3, 5:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 1* 1002 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS- -343- 4 RECORD(S) CANCELLED OR TRUNCATED - -342- 5 SELECTED RECORD(S) -		

Other LIMIT commands that would achieve the same results are

```
LIMIT 3, 5/CANCEL:
LIMIT 3, 5, 5:
LIMIT 3, 5/CANCEL, 5:
LIMIT 3, 5, 5/ALTFILE:
LIMIT 3,5/CANCEL, 5/ALTFILE:
```

Note: ALTFILE is the DDname.

Example 3

LIMIT 3, 3/TRUNCATE:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 1* 1002 1* 1003	
PR C1, OB C1 WH C1 LE 1005:	5	-340- 5 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 3 SELECTED RECORD(S) -	1* 1001 1* 1002 1* 1003	

Other LIMIT commands that would achieve the same results are

```
LIMIT 3, 3/TRUNCATE, 3:
LIMIT 3, 3/TRUNCATE, 3/ALTFILE:
```

Note: ALTFILE is the DDname.

Example 4

LIMIT 3, 5/TRUNCATE:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 1* 1002 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS - -343- 4 RECORD(S) CANCELLED OR TRUNCATED -342- 5 SELECTED RECORD(S) -	1* 1001 . . . 1* 1007	

Other LIMIT commands that would achieve the same results are

```
LIMIT 3, 5/TRUNCATE, 5:
LIMIT 3, 5/TRUNCATE, 5/ALTFILE:
```

Note: ALTFILE is the DDname.

Example 5

LIMIT 3, 7, 5:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 . . . 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1006:	6	1* 1001 . . . 1* 1005	1* 1001 . . . 1* 1006	
PR C1, OB C1 WH C1 LE 1007:	7	1* 1001 . . . 1* 1005	1* 1001 . . . 1* 1007	
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS - -343- 4 RECORD(S) CANCELLED OR TRUNCATED -342- 5 SELECTED RECORD(S) -	1* 1001 1* 1005 .	

Another LIMIT command that would achieve the same results is

```
LIMIT 3, 7/CANCEL, 5:
```

Note: ALTFILE is the DDname.

Example 6

LIMIT 3, 7, 5/ALTFILE:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 . . . 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1006:	6	1* 1001 . . . 1* 1005		1* 1001 . . . 1* 1006
PR C1, OB C1 WH C1 LE 1007:	7	1* 1001 . . . 1* 1005		1* 1001 . . . 1* 1007
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS - -343- 9 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		

Note: ALTFILE is the DDname.

Example 7

LIMIT 3, 7/TRUNCATE, 5:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 . . . 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1006:	6	1* 1001 . . . 1* 1005	1* 1001 . . . 1* 1006	
PR C1, OB C1 WH C1 LE 1007:	7	1* 1001 . . . 1* 1005	1* 1001 . . . 1* 1007	
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS - -343- 4 RECORD(S) CANCELLED OR TRUNCATED - -342- 5 SELECTED RECORD(S) - 1* 1001 . . . 1* 1005	1* 1001 . . . 1* 1007	

Example 8

LIMIT 3, 7/TRUNCATE, 5/ALTFILE:

Command	No. of Selected Records	Message File	Report File	Alt File
PR C1, OB C1 WH C1 LE 1002:	2	-340- 2 TOTAL SELECTED RECORDS - -343- 2 RECORD(S) CANCELLED OR TRUNCATED - -342- 0 SELECTED RECORD(S) -		
PR C1, OB C1 WH C1 LE 1003:	3		1* 1001 . . . 1* 1003	
PR C1, OB C1 WH C1 LE 1004:	4		1* 1001 . . . 1* 1004	
PR C1, OB C1 WH C1 LE 1005:	5		1* 1001 . . . 1* 1005	
PR C1, OB C1 WH C1 LE 1006:	6	1* 1001 . . . 1* 1005	. . .	1* 1001 1* 1006
PR C1, OB C1 WH C1 LE 1007:	7	1* 1001 . . . 1* 1005	. . .	1* 1001 1* 1007
PR C1, OB C1 WH C1 LE 1009:	9	-340- 9 TOTAL SELECTED RECORDS - -343- 4 RECORD(S) CANCELLED OR TRUNCATED - -342- 5 SELECTED RECORD(S) - 1* 1001 . . . 1* 1005	. . .	1* 1001 1* 1007

CONTROLLING EXECUTION IN EVENT OF ERRORS: STOP/CONTINUE

The STOP commands allow you to specify an upper limit of errors in a session. CONTINUE commands countermand STOP commands within a session.

There are two independent forms of the STOP command; each has a corresponding CONTINUE countermand. You can issue both forms in the QUEST language. Since the second form, STOP AFTER SCAN IF, affects the DEFINE language commands, but not QUEST commands, it is discussed in more detail in *SYSTEM 2000 DEFINE Language, Version 12, First Edition*.

The STOP IF command affects QUEST processing in batch jobs, but not in interactive sessions.

When you issue a STOP IF command, an error counter is set to zero. This counter is incremented once for each retrieval and update command in error or, if so specified, only for each update command in error. If the maximum error count specified in the command is reached, the following message is issued, followed by an exit from the software:

```
-370-  ERROR LIMIT EXCEEDED -
```

The error counter and the maximum error count are not altered when you change processors.

The STOP AFTER SCAN IF command uses a different, independent error counter and maximum error count.

The CONTINUE IF command cancels both the maximum error count and the keeping of the error counter.

If the STOP IF command was issued, it is reset by one of the following:

- an end-of-job, in which case a new job finds no limits to the number or type of command errors
- a modified STOP IF command
- a CONTINUE IF command.

Formats

```
STOP [AFTER SCAN] IF  |[ANY]  [UPDATE]  |COMMAND IS    REJECTED  :
                      |[n]               |COMMANDS ARE

CONTINUE [AFTER SCAN] IF  |COMMAND IS    REJECTED  :
                          |COMMANDS ARE
```

ANY is the default maximum error count, which specifies 1. That is, the following commands are equivalent:

```
STOP IF ANY COMMAND IS REJECTED:
STOP IF 1 COMMAND IS REJECTED:
STOP IF COMMANDS ARE REJECTED:
```

n is the maximum error count. The number must be greater than 0.

UPDATE If you omit the UPDATE option, all command errors are counted.

Considerations

- STOP IF remains in effect, even if you change processors.
- STOP AFTER SCAN also remains in effect when you change processors.
- The STOP commands remain in effect if you enter the CONTROL processor; however, errors in the CONTROL language do not alter the error counter.

Examples

The following examples show the interaction of STOP IF commands and CONTINUE IF commands. For purposes of clarification, the number of errors in the counter and the maximum error count are shown.

Example 1

```
                                        Number    Maximum
                                          in       Error
                                       Counter     Count

STOP IF 2 COMMANDS ARE REJECTED:          0         2
   retrieval or update error              1         2
STOP IF 3 COMMANDS ARE REJECTED:          0         3
   retrieval or update error              1         3
   retrieval or update error              2         3
   retrieval or update error              3         3
-370- ERROR LIMIT EXCEEDED -
03/02/90    16:39:26    END SYSTEM 2000 - RELEASE xx.x
```

Example 2

```
                                        Number    Maximum
                                          in       Error
                                       Counter     Count

STOP IF 2 COMMANDS ARE REJECTED:          0         2
   retrieval or update error              1         2
CONTINUE IF COMMANDS ARE REJECTED:        0         0
   retrieval or update error              0         0
STOP IF ANY COMMAND IS  REJECTED:         0         1
   retrieval or update error              1         1
-370- ERROR LIMIT EXCEEDED -
03/02/90    16:39:26    END SYSTEM 2000 - RELEASE xx.x
```

Example 3

```
                                        Number    Maximum
                                          in       Error
                                       Counter     Count

STOP IF 2 COMMANDS ARE REJECTED:          0         2
   retrieval or update error              1         2
STOP IF 2 UPDATE COMMANDS ARE REJECTED:   0         2
   retrieval error                        0         2
   update error                           1         2
   update error                           2         2
-370- ERROR LIMIT EXCEEDED -
03/02/90    16:39:26    END SYSTEM 2000 - RELEASE xx.x
```

PREVENTING EXECUTION OF COMMANDS: ENABLE/DISABLE EXECUTION

The ENABLE EXECUTION and DISABLE EXECUTION commands allow the software to scan retrieval and update commands while either enabling or disabling their execution.

- The ENABLE EXECUTION command allows each syntactically correct command to be executed.
- The DISABLE EXECUTION command prevents the execution of all update and retrieval commands.

After the software scans a retrieval or an update command for syntax errors, it checks an edit flag. If the flag is on (ENABLE), the software executes the command. If the flag is off (DISABLE), the software stops processing the command, although it continues to search for syntax errors.

The flag is on by default at the beginning of each session. It remains on until you issue the DISABLE EXECUTION command, even if you change processors. If you issue the DISABLE EXECUTION command, that command remains in effect until you issue the ENABLE EXECUTION command, even if you change processors.

The following QUEST commands are not executed when the flag is off (DISABLE EXECUTION):

ADD	COLLECT	LIST	REMOVE TREE
ASSIGN	DESCRIBE	LOAD	TALLY
ASSIGN TREE	DITTO	PRINT	UNLOAD
CHANGE	INSERT TREE	REMOVE	

Formats

```
ENABLE EXECUTION:
```

```
DISABLE EXECUTION:
```

Considerations

- You can also issue the ENABLE/DISABLE EXECUTION commands in the DEFINE language. If you issue the DEFINE language MAP command when the flag is disabled, neither the MAP command nor any subsequent QUEST language commands are executed. (See the ENABLE/DISABLE EXECUTION commands in *SYSTEM 2000 DEFINE Language, Version 12, First Edition* for more information.)
- The internal counters set by STOP IF and CONTINUE commands are not affected by the ENABLE/DISABLE EXECUTION commands.
- If DISABLE EXECUTION is issued before the LOAD command, the Data File is scanned for syntax errors, but the LOAD command is not executed.

MOVING UPDATED PAGES TO DISK: CLEAR

The CLEAR commands are available as an additional protection of the integrity of the database tables during update operations. These commands allow you to specify when modified database pages and/or Update Log pages are to be written from main memory to disk.

To alter a database, the pages of the database tables that need modification must be brought from disk to main memory. Ordinarily, an updated page is written back onto the database disk files by the buffer manager when the main memory occupied by that page must be made available for other processing.

QUEST provides four forms of commands to give you control over clearing updated database pages from memory to disk (see **Formats** on page 31-21). These commands also allow you to control the clearing of buffers that hold information to be written to an active Update Log.

The CLEAR commands not only clear pages from memory to disk, but they also set the mode of clearing, as described below.

Damage flags The software maintains an internal damage flag to indicate whether updated pages have been written back to disk. The flag shows a "damaged" status as long as the modified pages are in main memory and have not been written back to disk. It is marked "clean" after the pages are written to disk.

There is also a main memory damage flag that indicates the status of the database residing on disk and in main memory. This flag indicates "damaged" only during actual updating. During a session, a message is issued when a database becomes damaged and at other times, such as when a damaged database is opened or closed and when updates are attempted.

Effects of CLEAR commands Any CLEAR command produces both an immediate result and a continuing effect.

- The immediate result of CLEAR or CLEAR AUTOMATICALLY is the clearing of all modified database pages and all Update Log buffers. If the keywords UPDATE LOG are specified in either command (that is, CLEAR UPDATE LOG or CLEAR UPDATE LOG AUTOMATICALLY), only Update Log buffers are cleared to the Update Log disk file.

- The continuing effect of any CLEAR command is to set the mode of clearing for subsequent updates in the session. That clear mode remains in effect until you either issue another CLEAR command or exit the software.

  User-specified clear mode is set when a Self-Contained Facility user or a PLEX program issues a CLEAR or CLEAR UPDATE LOG command at specific intervals in a session.

  Automatic clear mode is set when a CLEAR AUTOMATICALLY or CLEAR UPDATE LOG AUTOMATICALLY command has been issued, and the software automatically clears all buffers or just the Update Log buffers after each update operation.

The CLEAR commands affect only the user that issues them. If an opened database does not have an active Update Log, a message is issued. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for complete discussions of all messages.

The software also clears buffers on its own in order to obtain buffer space during processing. Also, the software ensures that all buffers are cleared to disk after major operations, such as RESTORE, RELOAD, and CREATE/REMOVE INDEX, and before major processes, such as the SAVE command and the REPORT command. CLEAR commands issued by you clear buffers not cleared by these main processes and not cleared at the end of processing individual update commands.

The occasional use of a CLEAR or CLEAR UPDATE LOG command entails less I/O than clearing all buffers in automatic clear mode. Also, clearing only the Update Log (File 7) automatically causes no I/O with regard to modified database table pages and ensures that the database can be recovered in the event of a system failure.

The following charts summarize the immediate results and continuing effects of CLEAR commands. Update Log buffers are cleared to the Update Log.

Illus. 31.1 Immediate Results and Continuing Effects of CLEAR Commands

	Memory-Resident Modified Database Pages	
Command	**Immediate Results**	**Continuing Results**
CLEAR:	written to disk	user-specified clear mode
CLEAR UPDATE LOG:	none	user-specified clear mode
CLEAR AUTOMATICALLY:	written to disk	automatic clear mode
CLEAR UPDATE LOG AUTO:	none	user-specified clear mode

	Memory-Resident Update Log Pages	
Command	**Immediate Results**	**Continuing Results**
CLEAR:	written to disk	user-specified clear mode
CLEAR UPDATE LOG:	written to disk	user-specified clear mode
CLEAR AUTOMATICALLY:	written to disk	automatic clear mode
CLEAR UPDATE LOG AUTO:	written to disk	automatic clear mode

Formats

```
CLEAR:

[CLEAR] AUTOMATICALLY:

CLEAR UPDATE LOG:

[CLEAR] UPDATE LOG AUTOMATICALLY:
```

If you do not issue a CLEAR command, clearing is done by the buffer manager when space occupied by the updated database pages and/or Update Log pages needs to be made available, and by the software when certain commands are issued.

Considerations

- Abnormal job termination (for instance, telephone disconnect, system error, or time limit) can be trapped by the software as described in *SYSTEM 2000 Messages and Codes, Version 12, First Edition*. The following situations are possible:

    a) For a database with Rollback enabled, any CLEAR command issued between synchpoints or within a frame has no effect other than to change the clearing made if Rollback is disabled subsequently in the session.

    b) A trapped abnormal termination of a session in a single-user environment clears database pages and Update Log pages from main memory to disk. Damage occurs only if the session terminates during an update.

    c) In a single-user environment, if database pages in main memory have been updated but have not yet been cleared to disk before an untrapped termination of a session, the database on disk is in a damaged state. Frequent clearing of database pages reduces the chance of having a damaged database but incurs I/O. Clearing Update Log pages involves less I/O and affords recovery through the APPLY command.

- In a Multi-User environment, CLEAR does not guarantee a "clean" database since other users can be updating the database.

Abbreviations: AUTOMATICALLY = AUTO

- To protect against damage from an untrapped abnormal termination, the software clears the database pages and Update Log pages as a result of the execution of certain commands. Only the database associated with one of those commands is affected by the clearing. The software assures a "clean" database in the following situations:

  a) when you issue these commands:
     LOAD
     REPORT
     FRAME
     END FRAME

  b) after completing these commands:
     APPLY
     CREATE INDEX
     MAP
     RELOAD
     REMOVE INDEX
     REORGANIZE
     RESTORE

- When a CLEAR command is issued, its continuing effects remain until either another CLEAR command is issued or the job ends, even if you change databases within the session. See Illus. 31.1 on page 31-20 for the continuing effects of each CLEAR command.

Example

The following example shows the immediate results and continuing effects of CLEAR commands. Regardless of the clearing mode that is currently in effect, the software always clears modified database pages and Update Log buffers if more space is required during update processing. Also, all CLEAR commands and their continuing effects pertain to all databases currently open during the execution of a PLEX program. (For the Self-Contained Facility user, only one database is open at a time.)

Begin Job	
(update commands)	*Clearing done by buffer manager when the space occupied by the modified database pages or Update Log buffers are needed for other pages (default).*
`CLEAR UPDATE LOG AUTO:`	*Clears Update Log buffers only.*
(update command)	*Clears Update Log buffers only.*
(update command)	*Clears Update Log buffers only.*
`CLEAR:`	*Clears database pages and Update Log buffers.*
(update command)	*Clears nothing unless space is needed.*
`CLEAR UPDATE LOG:`	*Clears Update Log buffers only.*
(update command)	*Clears nothing unless space is needed.*
`CLEAR AUTO:`	*Clears database pages and Update Log buffers.*
(update command)	*Clears database pages and Update Log buffers.*
`CLEAR UPDATE LOG AUTO:`	*Clears Update Log buffers only.*
(update command)	*Clears Update Log buffers only.*
End of Job	*Clears database pages and Update Log buffers.*

Chapter 32

Repeating Command Units: DITTO, PREVIOUS, SAME

INTRODUCTION 32-2

REPEATING ACTION-CLAUSES: DITTO 32-2
Format 32-3
Examples 32-3

REPEATING VALUE STREAMS: PREVIOUS 32-4
Example 32-4

REPEATING QUALIFICATION CRITERIA: SAME 32-5
Considerations 32-5
Examples 32-6

SPECIFYING WHICH WHERE-CLAUSE TO REPEAT: SAME IS DYNAMIC/STATIC 32-7
Formats 32-7
Examples 32-8

MULTI-USER ENVIRONMENT CONSIDERATIONS FOR REPETITIVE PROCESSING 32-9
Self-Contained Facility $LOCK Option, Non-TP Users 32-9
Self-Contained Facility $LOCK Option, TP Users 32-9

INTRODUCTION

The operators discussed in this chapter offer you the convenience of repetitive processing.

- The **DITTO** operator repeats the action-clause of the most recently executed retrieval or update command.
- The **PREVIOUS** operator repeats the value stream contained in the action-clause of the most recently executed update command.
- The **SAME** operator retains and uses the list of qualified data records from a previously issued where-clause of a retrieval or update command.

REPEATING ACTION-CLAUSES: DITTO

DITTO repeats the most recently executed action-clause of a retrieval or update command. That is, DITTO repeats all of the previous command except its where-clause. If the previous command had no where-clause, DITTO simply repeats the command.

Using DITTO means you do not spend time repeating a command, and you do not risk making errors in the repetition.

You cannot use DITTO unless it is preceded by a retrieval or update command beginning with one of the following action verbs:

DESCRIBE	ASSIGN
LIST	CHANGE
PRINT	REMOVE
TALLY	ASSIGN TREE
UNLOAD	INSERT TREE
COLLECT	MOVE TREE
ADD	REMOVE TREE

If the preceding retrieval or update command contains a where-clause (after-clause or before-clause), the command with DITTO must have a where-clause with the same identifying keyword (that is, WHERE, AFTER, or BEFORE), although the conditions may qualify the same set or a different set of data records. Thus, DITTO allows you to do the same action on the same (or different) selected data records.

If the preceding retrieval or update command contains no where-clause, the command with DITTO must not include a where-clause.

Unless there has been a change of processor, DITTO repeats the action-clause of the previous command, beginning with one of the action verbs listed above, even if other QUEST or system-wide commands intervene, such as LIMIT, END LIMIT, or REPORT FILE IS.

If the action-clause being repeated contains an ordering-clause, DITTO repeats the ordering-clause also. If the action-clause being repeated does not contain an ordering-clause, you cannot add one to the DITTO command.

For a discussion of the DITTO operator in a Multi-User environment, see **Multi-User Environment Considerations for Repetitive Processing** on page 32-9.

Format

```
DITTO [where-clause] :
```

where-clause selects specific logical entries. See Chapter 24, "The Where-Clause," for details.

Examples

The following examples are general situations for which using DITTO would be advantageous.

Example 1

DITTO would be useful for performing the same update in different parts of the database, as described by different where-clauses. For example, in a library database, you could use DITTO to enter a date for all books returned on a given date.

```
ADD DATE RETURNED EQ 05/31/1990* WH TITLE EQ GONE WITH THE WIND:

DITTO WHERE TITLE EQ ROOTS:
```

Subsequent commands with DITTO could be issued, changing only the value for TITLE, until all the books returned on May 31, 1990, have had a value entered for DATE RETURNED.

Example 2

DITTO would also be useful for retrieving the same listings for different collections of selected data records.

For example, in a personnel database, you might need to produce a listing for each department of its employees' identification numbers, last names, and position titles. The following commands would produce these listings with the same format for each.

```
LIST/TITLE R(6) ID+NUMBER, L(10)LAST NAME, POSITION TITLE/C1, C2,
   C101 WHERE DEPARTMENT EQ EXECUTIVE:

DITTO WHERE DEPARTMENT EQ MARKETING:
```

You could continue with more DITTO commands, changing only the value for DEPARTMENT, until you have obtained listings for all departments.

REPEATING VALUE STREAMS: PREVIOUS

PREVIOUS repeats the value stream or item value given in the immediately preceding update command.

PREVIOUS is a convenience for specific kinds of repetitive update commands. Like DITTO, the primary advantage of PREVIOUS is that it reduces the time spent entering an update command and avoids the likelihood of making an error in the repetition.

You can use PREVIOUS after the keyword EQ in action-clauses for the following update commands:

ADD
ASSIGN
CHANGE
ASSIGN TREE
INSERT TREE

PREVIOUS causes the software to reuse the value stream found in an immediately preceding update command, including the END* (entry terminator). In the case where PREVIOUS repeats a single value in a preceding update command, the value and the asterisk following the value is reused by PREVIOUS. So, do not put an asterisk after the PREVIOUS operator.

The preceding command must use EQ in the action-clause. In addition, the binary operator EQ must precede PREVIOUS. Also, the item or record preceding the EQ operator must be the same as in the original command.

PREVIOUS cannot refer to a command that preceded a change of processors. That is, if you call another processor and then return to QUEST, you cannot use PREVIOUS until after a new update command is executed.

The command referenced by PREVIOUS must contain no errors.

The value stream to which PREVIOUS refers must be meaningful for the command in which it is used. That is, if the original command is either ASSIGN TREE or INSERT TREE and the value stream includes values for more than one data record, the command using PREVIOUS must also be either ASSIGN TREE or INSERT TREE. However, if the update command includes a value stream containing values for only the record being updated, the command using PREVIOUS can be any update command (except REMOVE TREE or REMOVE) that updates the same type of record.

For a discussion of the PREVIOUS operator in a Multi-User environment, see **Multi-User Environment Considerations for Repetitive Processing** on page 32-9.

Example

The following example inserts a new C120 record in the EMPLOYEE database for two different employees. The values for the record are the same for both employees.

```
IT C120 *1 EQ 121*06/29/1991*122*168.00*124*$1800.00*125*$178.80*
    126*$1547.64*END* WH C1 EQ 1133:

IT C120 *1 EQ PREVIOUS WH C1 EQ 1247:
```

REPEATING QUALIFICATION CRITERIA: SAME

SAME retains and uses the list of qualified data records from a previously executed retrieval or update command. The SAME operator

- allows you to repeat the where-clause of a previous command
- can be used with or without additional qualification criteria.

Like DITTO, SAME is a convenience to you in certain types of situations.

The primary advantage of SAME is that it reduces the time spent on where-clause processing by retaining the list of qualified data records resulting from a previous command, rather than returning to the database tables to find the same collection of data records again. In addition, it reduces the time spent entering a command and avoids the likelihood of making an error in the repetition.

At the beginning of each session, the SAME IS command is initialized to DYNAMIC, which causes the software to retain and use the list of data records qualified by the where-clause in the last executed command for the current database. The previous where-clause, to which SAME refers, may or may not contain SAME. In this way, the DYNAMIC state enables a sequence of commands to act on an evolving collection of data records. You can issue SAME IS STATIC to change the SAME operator from its initialized state of DYNAMIC to STATIC. See **Specifying Which Where-Clause to Repeat: SAME IS DYNAMIC/STATIC** on page 32-7 for information on the SAME IS command and the differences between DYNAMIC and STATIC.

Considerations

- You cannot use SAME in a where-clause if it is joined to an unindexed expression by the binary operator OR. If such a command is issued, the software issues the following error message:

```
-372- 'SAME' NOT ALLOWED IN NON-KEY CONTEXT-
```

- NOT SAME is allowed as long as it is not preceded by OR, although NOT usually implies non-key processing.
- Using SAME ignores qualified records that have been removed by REMOVE TREE commands subsequent to generation of the where-clause qualified list.
- In a Multi-User environment, other users changing the database may affect the list of qualified data records retained by the SAME operator. To use SAME, you may need to get exclusive use of the database. For a discussion of the SAME operator in a Multi-User environment, see **Multi-User Environment Considerations for Repetitive Processing** on page 32-9.

Examples

The following examples show some uses of the SAME operator. Each time SAME is used in a command, the equivalent where-clause is shown at the right.

Example 1

This example shows how SAME repeats the previous where-clause with additional qualification criteria, that is, using SAME IS DYNAMIC.

Commands	Equivalent Where-Clause
PR C1 WH C4 EQ 6 AND C9 LT 12:	
PR C5 WH SAME:	WH C4 EQ 6 AND C9 LT 12
PR C5 WH SAME AND C3 GT 8:	WH C4 EQ 6 AND C9 LT 12 AND C3 GT 8
PR C5 WH SAME:	WH C4 EQ 6 AND C9 LT 12 AND C3 GT 8

Example 2

This example shows the use of SAME when there are intervening commands with no qualification criteria between the command with the where-clause and the command with SAME.

Commands	Equivalent where-clause
PR C1 WH C3 GT 10:	
DESCRIBE:	
PR C0:	
TALLY C3:	
PR C2 WH SAME:	WH C3 GT 10

SPECIFYING WHICH WHERE-CLAUSE TO REPEAT: SAME IS DYNAMIC/STATIC

The SAME IS command tells the software which retained list of qualified data records the SAME operator is to use.

- In the STATIC state, the software retains and uses the same collection of qualified data records either alone or with additional qualification criteria.
- In the DYNAMIC state, the software acts on an evolving collection of data records.

The SAME operator is initialized to DYNAMIC at the beginning of each session. You can change the status of SAME at any time during a session.

In the DYNAMIC state, the software retains and uses the list of data records qualified by the last command with a where-clause that was executed. SAME may or may not be contained in the where-clause repeated by a subsequent SAME. This allows you to perform a series of actions on an evolving collection of data records. See Example 1.

Issuing the SAME IS STATIC command allows you to tell the software to retain and use the list of data records qualified by the more recent

- where-clause executed prior to the SAME IS STATIC command
- where-clause executed that does not contain SAME.

See Example 2.

The software produces unexpected results if you invoke a string when SAME is in the STATIC state and the invoked string includes several update or retrieval commands that assume SAME is in the DYNAMIC state. You can avoid unexpected results by displaying the string definition using the DESCRIBE command and then, if necessary, issuing SAME IS DYNAMIC.

Formats

```
SAME IS DYNAMIC :

SAME IS STATIC :
```

Abbreviations: SAME = SA

Examples

Example 1

Consider the following sequence of where-clauses when SAME is DYNAMIC. Those on the left make use of SAME while those on the right are the equivalent where-clause without SAME.

Specified Where-Clause	Equivalent Where-Clause
WH C7 EQ FEMALE	
WH SAME AND C102 EQ MARKETING	WH C7 EQ FEMALE AND C102 EQ MARKETING
WH SAME AND C111 GT 1500.00	WH C7 EQ FEMALE AND 102 EQ MARKETING AND C111 GT 1500.00

Compare these equivalent where-clauses with the ones below when SAME is set to STATIC.

Example 2

Consider the following sequence of where-clauses utilizing SAME and compare the equivalent where-clauses on the right with those above.

Specified Where-Clause	Equivalent Where-Clause
WH C7 EQ MALE	
SAME IS STATIC:	
WH SAME AND C102 EQ MARKETING	WH C7 EQ MALE AND C102 EQ MARKETING
WH SAME AND C111 GT 1500.00	WH C7 EQ MALE AND C111 GT 1500.00

MULTI-USER ENVIRONMENT CONSIDERATIONS FOR REPETITIVE PROCESSING

Self-Contained Facility $LOCK Option, Non-TP Users

When the $LOCK command is issued in a Self-Contained Facility job, it ties up one thread, but it does not require exclusive use of the database. Other Multi-User/Multi-Thread jobs in other threads can share a database.

The use of PREVIOUS requires the $LOCK command for both exclusive use and non-exclusive use of the database in the Multi-User/Multi-Thread environment because of scratch file allocation. The $LOCK command must be used to "frame" a segment consisting of commands that use PREVIOUS. The PREVIOUS operator is meaningful only within the segment in which it appears. At the beginning of the segment, PREVIOUS is not equivalent to any previous command syntax. It acquires meaning only after a retrieval or update command has been issued in the segment.

The use of DITTO does not require $LOCK if the DITTO execution parameter is set to YES. Likewise, $LOCK is not required for the SAME operator if the SAME execution parameter is set to YES. The default setting for both of these parameters is YES, which allows the use of DITTO and SAME across segments. For more details about these execution parameters, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*. Self-Contained Facility commands using the SAME operator ignore qualified data records that were removed by the same or another user.

Self-Contained Facility $LOCK Option, TP Users

Since the effect of a $LOCK command is ended at the end of a TP segment, the $LOCK command is unnecessary for TP users. A TP segment consists of a single Self-Contained Facility command or a series of commands containing up to 1200 characters. The TP segment begins with the first character transmitted and ends with the character that signals message completion to the host computer. Therefore, the TP user can use the PREVIOUS operator only when the following commands are issued in the same TP segment:

- the command containing the PREVIOUS operator
- the command to which PREVIOUS refers.

If the DITTO execution parameter is set to YES (default), you can use the DITTO operator across TP segments. Likewise, if the SAME execution parameter is set to YES (default), you can use the SAME operator across TP segments. For more details about these execution parameters, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

Self-Contained Facility commands using SAME ignore qualified data records that were removed by the same or another user.

Chapter 33

Obtaining Calculations: Ad Hoc Functions

Considerations 33-3
Examples 33-3

By using ad hoc functions within a command, you can obtain calculations with arithmetic expressions based on the data stored in your database. An ad hoc function can

- contain system functions and by-phrases
- contain the system string *FTODAY*
- contain stored functions
- contain numeric values and constants
- be nested
- be included in a by-clause.

You include ad hoc functions in the action-clause of retrieval and update commands specifying mathematical expressions to be performed on the data. The specified expressions are done once for each occurrence of the deepest schema item specified in the function.

You use ad hoc functions much like any component, except you must enclose the function in parentheses. You can also use pairs of parentheses in an ad hoc function to indicate the order of operation. The software first executes the expression indicated in the innermost pair of parentheses and works outward. Multiplication and division operations are done before addition and subtraction operations. If you specify both multiplication and division operations, the software does the calculations indicated on the left and works toward the right. (Example 4 shows the effect this could have on the results of INTEGER type ad hoc functions.)

The operands indicating each of these operations are shown below:

Operand	Operations
*	Multiplication
/	Division
+	Addition
-	Subtraction

Only numeric items in the same path can be included in ad hoc functions. If more than one item is included, the output format is determined by the highest item specified in the function, according to the order shown below, highest to lowest:

DOUBLE
REAL
DECIMAL
MONEY
INTEGER
DATE

Also, the software automatically converts different item types and constants to a common type when evaluating calculations. If an arithmetic operator, such as + or -, has operands of different types, the software converts the lower type to the higher type. The order of precedence from highest to lowest is DOUBLE, REAL, DECIMAL, MONEY, INTEGER, and DATE.

If the highest item is DECIMAL or MONEY, the output will be DECIMAL with three decimal places. If the highest item is INTEGER or DATE, the output will be INTEGER. In addition, you can include dates and constants in ad hoc functions. However, to avoid confusing the slashes with the division operand, you must specify dates using the form MM.DD.YYYY.

The output format of an ad hoc function containing MONEY items only is the MONEY format. The result has two places to the right of the decimal point and a dollar sign.

If you specify a system function, such as SUM, with a MONEY item and only one record is selected, no calculations are processed. The software prints the value as it is stored in the database.

If an ad hoc function contains a MONEY value and the decimal point is omitted, the software assumes the decimal point to be two places to the left of the last digit in the value, for example, PR ($1) yields $.01.

For example, if you want decimal output from an ad hoc function containing only INTEGER items, include a dummy decimal value in the function. For example, suppose C1=10 and C2=20. The ad hoc function (C1/C2*100) would give a result of zero because the software truncates 10/20=0.5 to 0. Note that the division operation is done before the multiplication operation since the software works from left to right in a mathematical subexpression when the operations indicated are multiplication and division. You can force decimals by including a dummy decimal in the ad hoc function as shown below.

```
((C1+0.0)/C2*100) = ((10+0.0)/20*100) = 0.5*100 = 50.
```

All ad hoc functions containing a DECIMAL item or value yield a result with three places to the right of the decimal point. MONEY items yield two places. However, if you issue an ad hoc function in an update command, for example, CHANGE C1=(C2+C3), and the items or constants involved have four or more digits to the right of the decimal point, the calculation retains only the first three digits to the right of the decimal point for type DECIMAL. For type MONEY, two digits are retained. The result is stored in the database that way.

You can specify component names in an ad hoc function as long as they do not include slashes, plus signs, or minus signs. For example, if the item QUANTITY-RECEIVED is to be used in an ad hoc function, the component number should be specified to avoid confusion with the minus sign.

You can also use system functions in ad hoc functions if each system function is enclosed in parentheses. All calculations are done at the time of retrieval, but the results are not stored.

You can store an ad hoc function in the database definition as a stored function by assigning a component name, number, and type during a DEFINE session. Storing the function is preferable when a complex mathematical expression is involved, and it is used often or by different users. By storing the function, you can also specify the type of output to be displayed (DOUBLE, REAL, DECIMAL, MONEY, INTEGER, or DATE), something not possible for ad hoc functions.

Considerations

- If one or more values for items specified in an ad hoc function is null, no output is produced unless the NULL or ZERO format option is in effect. If NULL is set, the output associated with the missing values is -NULL-. If ZERO is set, the missing values are replaced with zeros, and the calculations are made and displayed. However, nulls for dates are not converted to zero.

- Ad hoc functions can contain Collect File items when used in retrieval commands, but not in update commands.

- In ad hoc functions, dates are computed as the number of days since October 15, 1582, the date the Gregorian calendar became effective.

- The results of ad hoc functions are identified by a stub number, starting with 10001. That is, if you specify three ad hoc functions in a retrieval command, the output will be identified with stub numbers 10001, 10002, and 10003. (See the following examples.)

- You can specify constants as whole numbers, decimal numbers, or in E-notation (for example, 2.567E+7).

- UNDEFINED items are treated as text and cannot be used in functions.

- If a valid numeric constant conflicts with a component name, the constant is used.

Examples

The first four examples in this section use a hypothetical database. The three logical entries are shown below. Note that C1, C2, and C3 are INTEGER items; C4 and C5 are DECIMAL items; C6 and C7 are MONEY items; and C8 and C9 are dates.

```
1* 2              1* 4              1* 6
2* 21             2* 17             2* 13
3* 4              3* 5              3* 6
4* 7.0            4* 7.0            4* 7.0
5* 8.0            5* 9.0            5*
6* $21.25         6* $21.50         6* $22.00
7* $20.00         7* $25.00         7* $18.00
8* 05/20/1987     8* 05/15/1987     8* 05/10/1987
9* 07/10/1991     9* 07/15/1991     9* 07/20/1991
```

Example 1

This example shows several ad hoc functions in the same command. The first ad hoc function (10001* in the output) contains INTEGER items, the second (10002*) contains DECIMAL items, the third (10003*) contains MONEY items, and the fourth (10004*) contains dates.

Note that the results of the ad hoc functions 10001* and 10004* are displayed as integers, while the results of ad hoc functions 10002* and 10003* are both displayed as decimals with three places to the right of the decimal point.

```
PR ((C1+C2)/C3), (C4-C5), (C6+C7), (C9-C8):

10001*  5
10002* -1.000
10003* $41.25
10004*  1512
10001*  4
10002* -2.000
10003* $50.50
10004*  1522
10001*  3
10003* $40.00
10004*  1532
```

Example 2

This example uses different types of items in the same ad hoc function.

```
PR (C2+C4), (C3+C6), (C1+C5):

10001*  28.000
10002*  25.250
10003*  10.000
10001*  24.000
10002*  30.500
10003*  13.000
10001*  20.000
10002*  28.000
```

Example 3

This example shows how you can control the order in which the software does the calculations.

The first ad hoc function (10001* in the output) includes no parentheses to change the order of calculations. The second ad hoc function (10002*) has parentheses enclosing the last two items. The third ad hoc function (10003*) has parentheses enclosing the first two items and also the last two items.

```
PR (C1+C2*C3/C5-C4), (C1+C2*C3/(C5-C4)), ((C1+C2)*C3/(C5-C4)):

10001*  5.500
10002*  86.000
10003*  92.000
10001*  6.444
10002*  46.500
10003*  52.500
```

The table below shows the order in which the software does the indicated operations for each of the three ad hoc functions. Note that the results are different for each one.

(C1+C2*C3/C5-C4)	(C1+C2*C3/(C5-C4))	((C1+C2)*C3/(C5-C4))
C2*C3	C5-C4	C1+C2
[C2*C3]/C5	C2*C3	C5-C4
C1+ [C2*C3/C5]	[C2*C3]/[C5-C4)]	[C1+C2] *C3
[C1+C2*C3/C5]-C4	C1+[C2*C3/C5-C4]	[C1+C2*C3]/[C5-C4]

Example 4

This example shows how the order in which the software does calculations can affect the results of ad hoc functions containing only INTEGER items.

The first ad hoc function (10001* in the output) shows how the truncation of the decimal places can produce unexpected results. The second ad hoc function (10002*) shows using a dummy decimal in changing the function from INTEGER to DECIMAL.

```
PR (C1/C3*100), ((C1+0.0)/C3*100)):

10001*  0
10002*  .005
10001*  0
10002*  .008
10001*  100
10002*  .010
```

Example 5

This example uses the EMPLOYEE database to show the result of an ad hoc function containing items from different levels. Note that the software computes a value for each NET PAY (C126) value found under the record containing the CURRENT DEDUCTION (C114) for the employee with the EMPLOYEE NUMBER (C1) of 1010.

```
PR (C114 + C126) WH C1 EQ 1010:

10001* $2,250.00
10001* $2,250.00
10001* $2,250.00
10001* $2,250.00
10001* $2,250.00
```

Chapter 34

Obtaining Arithmetic Statistics: System Functions

USING SYSTEM FUNCTIONS 34-2
- *COUNT* 34-2
- *SUM* 34-2
- *MIN and MAX* 34-2
- *AVG* 34-2
- *SIGMA* 34-3
- *Format* 34-3
- *Considerations* 34-4
- *Examples* 34-4

SPECIFYING THE FOCAL RECORD FOR A SYSTEM FUNCTION 34-6
- *Considerations* 34-6
- *Examples* 34-7

System functions allow you to obtain simple arithmetic statistics about the values stored in a database. The system functions are: AVG, COUNT, MAX, MIN, SIGMA, and SUM.

You can use system functions

- in the action-clause of LIST, PRINT, and UNLOAD commands. (You can use more than one system function in a single command.)
- in an ordering-clause with a by-phrase.
- in update commands and ad hoc functions, if they are enclosed in parentheses.
- in a command containing a where-clause (but not in the where-clause).
- in stored functions.

When you execute a command with a system function, the output for the system function is displayed with a function label.

USING SYSTEM FUNCTIONS

COUNT

The COUNT system function produces a count of the number of times a value exists or a data record occurs in the database. Counting record occurrences does a minimum amount of I/O (with or without a where-clause). Counting item values requires referencing both the Hierarchical Table and the Data Table to exclude nulls from the count. COUNT is the only system function that can be used with data records. The result of COUNT is equivalent to *n* in statistical equations.

SUM

The SUM system function, usable with numeric items only, adds the values for the requested item. The single number produced contains three decimal places for DECIMAL items, two places plus a dollar sign for MONEY items, and no decimal places for DATE and INTEGER items. DOUBLE and REAL items produce E-notation output. (Dates are treated as the number of days that have elapsed since October 10, 1582.) The SUM system function can be expressed as

$$\sum_{i=1}^{i=n} X_i$$

where $\sum$ is the summation symbol and X_i is each of the object values in the selected records, from the first one to the *n*th one.

MIN and MAX

You can use the MIN and MAX system functions with either alphabetic or numeric items. The MIN system function produces the minimum value stored for the requested item, and the MAX system function produces the maximum value stored for the requested item. For CHARACTER, TEXT, and UNDEFINED items, MIN and MAX use the collating sequence shown in Appendix B.

AVG

The AVG system function produces the average of the data values stored in the database for the requested numeric item. It uses the equation

$$AVG = \sum_{i=1}^{i=n} \frac{X_i}{n}$$

where

$\sum$ is the summation symbol.

X_i is each of the values for the requested item in the selected records, from the first one to the *n*th one.

n is the number, or count, of the values that were added together.

You can use AVG with numeric items only.

SIGMA

The SIGMA system function produces the standard deviation for the requested numeric item. The formula for the standard deviation is

the square root of $\frac{1}{n-1}\left[\sum_{i=1}^{i=n} X_i^2 - \frac{1}{n}\left(\sum_{i=1}^{i=n} X_i\right)^2\right]$

You can use SIGMA with numeric items only.

Format

```
|AVG     |unit                    [by-phrase]
|COUNT
|MAX
|MIN
|SIGMA
|SUM
```

unit

```
|item
|record
|stored-function
|(adhoc-function)
|cf-item
|(system-function)
```

item is a schema item name or component number.

record is a schema record name or component number.

stored-function is a stored function name or component number with optional parameters.

adhoc-function is a mathematical expression enclosed in parentheses. See Chapter 33, "Obtaining Calculations: Ad Hoc Functions" for details.

cf-item is a Collect File item name.

system-function is a system function enclosed in parentheses.

by-phrase

```
BY |record
   |DATA BASE
```

With a system function, the by-phrase must contain an ancestor record name or number. This capability provides a great deal of flexibility.

For example, issuing the command shown below produces the sum of GROSS PAY values stored in the EMPLOYEE database for each entry. This involves a single value for each employee instead of a single value for the entire database.

```
PR SUM C124 BY C0:
```

See **Specifying the Focal Record for a System Function** on page 34-6 for details.

Considerations

- You cannot use Collect File items with system functions.
- COUNT is the only system function you can use with records.
- A system function can have only one unit or system function.
- Nulls are treated as zero only when the ZERO format option is in effect and when the object of a system function is enclosed in parentheses, for example, SUM (C5).
- If you use a system function in an ad hoc function, enclose the system function in parentheses. For example,

```
PR (((MIN C126) + (MAX C126))/2):
```

- When SUM, AVG, or SIGMA is used with a MONEY item, the output is a decimal number with three decimal places; that is, the output includes no dollar sign or CR. The output of REAL and DOUBLE is E-notation.
- Without a where-clause, any system functions that neither contain a by-phrase nor are contained within one are processed first. Each system function requires one logical order scan of the database.

Examples

The following examples use the EMPLOYEE database. In each case, the default format options ZERO SUPPRESS and NULL SUPPRESS are set.

Example 1

This example shows the MIN and MAX system functions used with a CHARACTER item (LAST NAME). Note that the MIN and MAX for alphabetic items is based on alphabetical order.

```
PR MIN C2, MAX C2:

MIN 2* AMEER
MAX 2* WILLIAMSON
```

Example 2

This example uses each system function with the same numeric item (NET PAY). Note that each system function requires a separate logical order scan of the database and produces one value.

```
PR COUNT C126, MIN C126, MAX C126, SUM C126, AVG C126, SIGMA C126:

CNT 126*  238
MIN 126*     $364.27
MAX 126*   $7,650.30
SUM 126* $590,030.39
AVG 126* $2,479.12
SIG 126*  1493.612
```

Example 3

This example uses SUM followed by a by-phrase. The where-clause is included to limit the output since the by-phrase instructs the software to compute the sum of all values stored in GROSS PAY for each entry. In other words, the software computes one value for each C0 record or entry.

```
PR SUM C124 BY C0 WH C1 LE 1010:

SUM 124* $39,800.00
SUM 124* $7,500.00
SUM 124* $27,000.00
SUM 124* $24,750.00
SUM 124* $6,000.00
SUM 124* $21,750.00
SUM 124* $11,250.00
SUM 124* $19,500.00
SUM 124* $11,250.00
SUM 124* $17,250.00
```

Example 4

This example uses a system function that has a system function as its object. In this case, the software computes the sum of the values in GROSS PAY for each entry and then computes the average of all the SUMs, yielding a single value.

```
PR AVG (SUM C124 BY C0):

AVG 10001* $14,167.89
```

SPECIFYING THE FOCAL RECORD FOR A SYSTEM FUNCTION

The by-phrase allows you to specify the level at which a system function is to be computed. That is, the by-phrase allows you to specify the focal record for a system function.

By including a by-phrase with a system function, you tell the software the focal record for processing the system function.

If you do not specify a focal record, the software uses the items from all selected records in the database to compute the result of the system function. In the EMPLOYEE database, you could use a by-phrase to compute the total GROSS PAY for an individual. That is, if you specify BY ENTRY, the total GROSS PAY is computed for each entry containing a selected record. If you do not include a by-phrase, one total would be computed for all employees in the database.

If you specify an ancestor of a qualified record to be the focal record, the software includes additional records in that family that would not be eligible with a lower focal record. (See Example 1.)

If you specify BY DATA BASE, the software processes the system function for the entire database, even if the command contains a where-clause. If the command contains no by-phrase and no where-clause, the software processes the system function as though BY DATA BASE had been specified.

Considerations

- The record specified in a by-phrase must be an ancestor of the object of the system function.
- When the focal record is raised, all its descendants become eligible to be selected records.
- When a by-phrase is used in a by-clause, the by-phrase overrides the by-clause in processing the system function with which it is used.
- If one or more retrieval objects are within a by-clause in a COLLECT command, all retrieval objects outside that by-clause will have nulls in the record when a Collect File record is written.

Examples

These examples use the EMPLOYEE database. The where-clause included in each command qualifies only data records from Logical Entry Two, which is shown at the end of this manual.

Note that Logical Entry Two has two C110 records, with two C120 records under one C110 record and three C120 records under the other. The command below contains the system functions used in the examples, but it does not contain a by-phrase. It is shown for comparison. Here, the sums pertain to all five C120 records in this logical entry.

```
PR SUM C124, SUM C126 WH C1 EQ 1120:

SUM 124* $9,450.00
SUM 126* $8,300.25
```

The first value is the sum of the five C124 values, and the second value is the sum of the five C126 values.

Example 1

This example shows two uses of a system function followed by a by-phrase. Note that the result of the second one listed, SUM C124 BY DATA BASE, is displayed before the results of the first one, SUM C124 BY C110.

```
PR SUM C124 BY C110, SUM C124 BY DATA BASE WH C1 EQ 1120:

SUM 124* $708,394.42
SUM 124* $5,850.00
SUM 124* $3,600.00
```

The software processes system functions with the by-phrase BY DATA BASE first, whether the BY DATA BASE is specified or implied. Including BY DATA BASE in this case obtains a value for the entire database, even though a where-clause is included in the command. The first system function with a by-phrase, SUM C124 BY C110, yields two values -- the first value is the sum of the three C124 values under record number 38, and the second value is the sum of the two C124 values under record number 14.

Example 2

This example includes two similar by-clauses, the only difference being the record by which the system functions are to be computed. Note that when two by-phrases are included in the same command, the specified records (for example, C110 and C100) must be related.

The selected data records are 38, 14, 15, and 17. For each selected record, the software processes the retrieval-clause from left to right:

- For record 38, the sums are calculated from the C120 records under it and then from the C120 records under its C100 parent (record 13).
- For record 14, the sums are calculated from the C120 records under it and then from the C120 records under its C100 parent (record 13).
- For record 15, since there are no C120 records under it, the software calculates the sums from the C120 records under its C100 parent (record 13) only.
- For record 17, no calculation takes place; neither it nor its parent C100 record (16) have C120 descendants.

```
PR BY C110, SUM C124, SUM C126, BY C100, SUM C124, SUM C126
   WH C1 EQ 1120:

SUM 124* $5,850.00
SUM 126* $5,131.05
SUM 124* $9,450.00
SUM 126* $8,300.25
SUM 124* $3,600.00
SUM 126* $3,169.20
SUM 124* $9,450.00
SUM 126* $8,300.25
SUM 124* $9,450.00
SUM 126* $8,300.00
```

Example 3

This example shows a by-phrase and a by-clause in the action-clause. Note that no value for the system function SUM C124 is displayed under the EFFECTIVE DATE* 02/16/1988. This is because there is no C120 record (and thus no C124 values) under the C110 record with this value for the entry.

```
PR C2, C101, BY C100, C111, C113, SUM C124 BY C110 WH C1 EQ 1120:

2* REID
  101* PROGRAMMER
    111*   $1,950.00
    113* 03/01/1991
SUM 124* $5,850.00
    111*   $1,800.00
    113* 03/01/1990
SUM 124* $3,600.00
    111*   $1,650.00
    113* 03/01/1989
2* REID
  101* ASSISTANT PROGRAMMER
    111*   $1,500.00
    113* 02/16/1988
```

Chapter 35

Using Stored Strings and Functions

INTRODUCTION 35-1

INVOKING STORED SIMPLE STRINGS 35-2
Format 35-2
Considerations 35-2
Examples 35-4

INVOKING STORED PARAMETRIC STRINGS 35-6
Format 35-6
Considerations 35-6
Examples 35-7

INVOKING STORED SIMPLE FUNCTIONS 35-10
Format 35-10
Considerations 35-10
Examples 35-11

INVOKING STORED PARAMETRIC FUNCTIONS 35-12
Format 35-12
Considerations 35-12
Examples 35-12

INTRODUCTION

You can store strings and functions in a database definition. *Strings* store sequences of command syntax (part of a command, a complete command, or several commands). *Functions* store arithmetic expressions. When you invoke a stored string or function, the software executes the stored syntax.

Strings and functions can be simple or parametric. *Simple* strings and functions have no parameters and are invoked by their component names or numbers, set off by system separators. *Parametric* strings and functions contain parameters for which you supply values when the string or function is invoked.

You define strings and functions using the DEFINE language, which stores the syntax in the database definition. For information on defining strings and functions, see *SYSTEM 2000 DEFINE Language, Version 12, First Edition.*

You invoke strings and functions in QUEST, which is explained in the following sections.

INVOKING STORED SIMPLE STRINGS

A simple string can store part of a command, a complete command, or even several commands. A simple string can even call one or more other strings or functions. Since the software replaces the call for the string with the string's stored syntax, you must supply the rest of the command syntax when invoking a string that is a partial command.

You can invoke a simple string at the beginning of a command or after a keyword or symbol recognized by the software. However, you cannot invoke a simple string in the place of a format option specification or immediately following one.

You invoke a simple string by issuing the system separator, the string name or component number, then another system separator. For example,

```
*C2001*
```

Format

```
* component *
```

component is the component name or number (including the C) of the stored string.

Considerations

- If the software finds a syntax error, string expansion is inhibited. Therefore, any command delimiter, usually the colon, coded within a string will not be detected. Including a command delimiter at the end of each command that invokes a string will ensure that the software processes each command.

- For the following commands, arrows indicate where a string can be invoked. As the consecutive arrows show, strings can be invoked anywhere after a keyword or symbol. In other words, a string must be invoked at a position where the software recognizes it as a string call and not as part of a value or component name. Therefore, within the layout specifications for a LIST command, strings can be invoked only at the positions indicated by the arrows because blanks are significant in that context.

```
PRINT C1,  C2, C3, C102, OB C102,C2 WHERE C7 EQ FEMALE AND
   C105 GT 06/01/78 AT 1:
LIST /REPEAT SUPPRESS, ZERO, TITLE (40) D(3) REPORT HEADING,
   L(5)COL1+HEAD, R(6) COL2 HEAD,S(1,2,1), F(30)FOOTING/ C1,
   C11, OB C1, C11 WHERE C1 EXISTS:
CHANGE C120 *1*2 EQ 121*02/28/91*122*160.00*126*1584.60*END*
   WH C1 EQ 1120:
RT C120 WH C121 LT 01/01/91:
```

- Assume that two strings have been defined as follows:

```
1001*ABC (STRING $ WHERE C1 EXISTS $):
1002*DEF (STRING $ MONTHLY REPORT $):
```

If string ABC is invoked in a PRINT command,

```
PRINT C2 *ABC*:
```

the software tries to process the command by searching for component C2 *ABC*. Since that component does not exist, the software issues an error message. On the other hand, if string ABC is invoked in an ASSIGN command,

```
ASSIGN C9 EQ FULL TIME* *ABC*:
```

the software processes the command correctly because the asterisk after FULL TIME delimits the value specified, and the software can recognize *ABC* as a string call. If string DEF is invoked immediately after a D-specification in a LIST command,

```
... D(5)*DEF* ... :
```

the software recognizes *DEF* as a string call and processes the command correctly. However, if string *DEF* is invoked after one or more blank spaces,

```
... D(5)␣*DEF* ... :
```

the software does not recognize *DEF* as a string call and processes the command with ␣*DEF* as the title of the report.

Examples

The following examples use the EMPLOYEE database. For each example, the string definition precedes the command invoking the string.

Example 1

This example shows how to invoke a simple string that stores a complete command. Note that since the command terminator (the colon) is included as part of the string definition, the colon is not required when the string is invoked, although including it would not affect the results.

```
2001* STRNG1 (STRING $PR C1, C2, C3, C102, OB C102, C2 WH C7
   EQ FEMALE AND C105 GT 06/01/90 AT 1:$):
```

Since the string is a complete command, you invoke it by its component name or number without including the command terminator.

```
*C2001*

*STRNG1*
```

Example 2

This example shows how to invoke a simple string that contains two complete commands. The first command is the same as that used in Example 1. The second command uses the DITTO operator and changes the first condition in the where-clause. Note that the command terminator is again part of the string definition.

```
2002* STRNG2 (STRING $PR C1, C2, C3, C102, OB C102, C2 WH C7
   EQ FEMALE AND C105 GT 06/01/90 AT 1: DITTO WH C7
   EQ MALE AND C105 > 06/01/90 AT 1:$):
```

The string can be invoked by its component name or number:

```
*C2002*

*STRNG2*
```

Example 3

This example shows how to invoke a simple string that contains a partial command, the two conditions used in the previous two examples. By providing the action-clause and the keyword WHERE when the string is invoked, the results are the same as in Example 1 and the first part of Example 2. Notice that the command terminator is not part of the string definition, so it must be specified when the string is invoked.

```
2003* STRNG3 (STRING $C7 EQ FEMALE AND C105 GT 06/01/90 AT 1 $):
```

The string can be invoked as in the following command:

```
PR C1, C2, C3, C102, OB C102, C2 WHERE *C2003*:
```

Example 4

This example shows how to invoke a simple string that invokes another simple string. Here STRNG3 is the same as in Example 3, and STRNG4 is the action-clause plus the keyword WHERE used to invoke the string in Example 3.

```
2003* STRNG3 (STRING $C7 EQ FEMALE AND C105 GT 06/01/90 AT 1 $):

2004* STRNG4 (STRING $PR C1, C2, C3, C102, OB C102, C2 WH *STRNG3* $):
```

You can invoke the strings by issuing either of the following commands:

```
*C2004*:

*STRNG4*:
```

INVOKING STORED PARAMETRIC STRINGS

A parametric string, like a simple string, can store part of a command, a complete command, or several complete commands. A parametric string can also call one or more other strings or functions. The difference is that a parametric string includes one or more parameters for which you supply values at the time the string is invoked. Component names or numbers, Collect File item names (including the (CF) indicator), item values, by-phrases, or one or more conditions in a where-clause are examples of values that you can supply for parameters.

You invoke a parametric string by issuing the system separator, the string name or component number, and the appropriate values (enclosed in parentheses) for the parameters. For example,

```
*C3001(C1, JONES):
```

Format

```
* component (parameters)
```

component is the component name or number (including the C) of the stored string.

parameters are values for the numbered parameters that were defined in the string. The values you supply must be specified in the numeric order of the parameters, regardless of the order they appear in the string, and they must be separated by commas. Parameters must be enclosed in parentheses.

Considerations

- In the commands below, arrows indicate where a parametric string can be invoked. In general, parametric strings can be invoked in the same positions as simple strings.

```
PRINT C1, C2,C3,  C102, OB C102,C2 WHERE C7 EQ FEMALE AND C105
↑    ↑↑  ↑↑  ↑  ↑↑↑    ↑↑ ↑↑    ↑       ↑↑    ↑↑         ↑↑
   GT 06/01/90 AT 1:
     ↑↑          ↑↑
LIST /REPEAT SUPPRESS, ZERO, TITLE (40) D(3) REPORT HEADING,
↑   ↑↑                              ↑↑   ↑    ↑             ↑
   L(5)COL1+HEAD, R(6) COL2 HEAD,S(1,2,1), F(30)FOOTING/ C1,
     ↑ ↑    ↑    ↑    ↑            ↑ ↑ ↑  ↑  ↑  ↑       ↑↑  ↑
   C11, OB C1, C11 WHERE C1 EXISTS:
   ↑   ↑↑ ↑↑  ↑↑        ↑↑
CHANGE C120 *1*2 EQ 121*02/28/91*122*160.00*126*984.60*END*
↑     ↑↑     ↑ ↑   ↑    ↑            ↑          ↑
   WH C1 EQ 1120:
     ↑↑    ↑↑
RT C120 WH C121 LT 01/01/91:
↑ ↑↑      ↑↑      ↑↑
```

Examples

The following examples use the EMPLOYEE database. For each example, the string definition precedes the command invoking the string.

Example 1

This example shows invoking a parametric string that stores a partial command. The parametric string has three parameters:

- The first parameter requires a value for the component GENDER (C7).
- The second parameter requires a value for the component START DATE (C105).
- The third parameter requires a positive integer as the object of the AT operator.

Since the parametric string is a partial command, you must supply the rest of the command syntax, as well as the values for the parameters.

```
2001* PARASTRG1 (STRING $ C7 EQ *1* AND C105 GT *2* AT *3* $):
```

With the following command, this string can be invoked to produce the same output as the examples for the simple strings in the previous section:

```
PR C1, C2, C3, C102, OB C102, C2 WH *PARASTRG1(FEMALE, 06/01/90, 1):
```

This is equivalent to issuing the following command:

```
PR C1, C2, C3, C102, OB C102, C2 WH C7 EQ FEMALE AND C105
   GT 06/01/90 AT 1:
```

Example 2

This example shows invoking a parametric string that stores a complete command. The parametric string has four parameters, numbered out of sequence:

- The first parameter requires a retrieval keyword, such as LIST, PRINT, or UNLOAD.
- The second parameter is numbered 3 and can be replaced with a condition.
- The third parameter is numbered 2 and requires a value for the component START DATE (C105).
- The fourth parameter requires a positive integer as the object of the AT operator.

```
2002* PARASTRG2 (STRING $ *1* C1, C2, C3, C102, OB C102 WH *3*
   AND C105 EQ *2* AT *4* $):
```

When you invoke the string, you must specify the values for the parameters in numerical order, not in the order in which they appear in the string.

The following command produces results similar to those in Example 1. However, since EQ can be either a binary or ternary operator, a range of values for START DATE (C105) is specified in this example.

```
*PARASTRG2(PR, 06/01/90*04/30/91, C7 EQ FEMALE, 0):
```

This is equivalent to issuing the following command:

```
PR C1, C2, C3, C102, OB C102 WH C7 EQ FEMALE AND C105 EQ
   06/01/90* 04/30/91 AT 0:
```

Example 3

This example includes three strings. The first string is a simple string containing a partial command, the action-clause with title specifications for the LIST command. The second string is a parametric string that invokes the first string. The single parameter requires a value for DEPARTMENT (C102). The third string is a simple string containing four complete commands, each invoking the second string (C2004) but with different values supplied for the parameter.

```
2003* PARASTRG3 (STRING $ LI/TITLE LAST NAME, FIRST NAMES,
   R(8) EMPLOYEE+AGE , R(8) TENURE / C2, C3,
   ((*FTODAY*-C5)/365.25), ((*FTODAY*-C4)/365.25), OB C2 $):

2004* PARASTRG4 (STRING $ *PARASTRG3* WH C102 EQ *1*):

2005* PARASTRG5 (STRING $ *PARASTRG4 (/ADMINISTRATION
   & FINANCE/): *PARASTRG4 (EXECUTIVE): *PARASTRG4
   (INFORMATION SYSTEMS): *PARASTRG4 (MARKETING): $):
```

Invoking the third string produces four sets of results, one for each department in the database, as invoked with the parameter values in this string.

```
*PARASTRG5*:
```

This is equivalent to issuing the following commands:

```
LI/TITLE LAST NAME, FIRST NAMES, R(8)EMPLOYEE+AGE ,R(8) TENURE /
   C2, C3, ((*FTODAY*-C5)/365.25), ((*FTODAY*-C4)/365.25), OB C2 WH
   C102 EQ /ADMINISTRATION & FINANCE/:

LI/TITLE LAST NAME, FIRST NAMES, R(8)EMPLOYEE+AGE ,R(8) TENURE /
   C2, C3, ((*FTODAY*-C5)/365.25), ((*FTODAY*-C4)/365.25), OB C2 WH
   C102 EQ EXECUTIVE:

LI/TITLE LAST NAME, FIRST NAMES, R(8)EMPLOYEE+AGE ,R(8) TENURE /
   C2, C3, ((*FTODAY*-C5)/365.25), ((*FTODAY*-C4(/365.25), OB C2 WH
   C102 EQ INFORMATION SYSTEMS:

LI/TITLE LAST NAME, FIRST NAMES, R(8)EMPLOYEE+AGE ,R(8) TENURE /
   C2, C3, ((*FTODAY*-C5)/365.25), ((*FTODAY*-C4(/365.25), OB C2 WH
   C102 EQ MARKETING:
```

INVOKING STORED SIMPLE FUNCTIONS

Simple functions allow you to store mathematical expressions to be performed on the data. Stored functions provide a label for the output. They also allow you to specify the numeric data type of the output, that is, DOUBLE, REAL, DECIMAL, MONEY, INTEGER, or DATE. You invoke stored simple functions in the action-clause of retrieval and update commands.

You can store any ad hoc function in the definition as a stored simple function by assigning a component name, number, and numeric item type during a DEFINE session. (The default item type for a function is DECIMAL.)

Storing a function is preferable when a complex mathematical expression is used often or by different users. If the data type required in the output from a function is DOUBLE, REAL, MONEY, or DATE, the function must be stored and the appropriate numeric data type specified. If a function has any decimal or floating point values (components or constants) and the output required is INTEGER, the function must be specified as INTEGER. If all values in the function are INTEGER, the output is INTEGER.

You invoke a simple function by issuing the system separator, the function name or component number, followed by another system separator. For example,

```
*C2010*
```

Format

```
* function *
```

function is the component name or number (including the C) of the stored function.

Considerations

- If one or more values for items specified in a stored simple function are null, no output results unless the NULL or ZERO format option is in effect. If NULL has been set, the output associated with the missing values is -NULL-. If ZERO has been set, the missing values are replaced with zeros and the calculations are made and displayed. However, nulls for dates are not converted to zero.

- If both multiplication and division operations are specified in a stored function, the software does the calculations indicated on the left and works toward the right.

- In stored functions, dates are computed as the number of days since October 15, 1582, the date the Gregorian calendar became effective.

- For INTEGER functions, the software truncates decimal values. For DECIMAL and MONEY functions, the results are rounded to three or two decimal places, respectively, except as discussed below. REAL and DOUBLE functions use floating point single or double precision, respectively.

- If a DECIMAL or MONEY function is specified with INTEGER items only, all decimal values are truncated in the calculations, although the output includes three zeros to the right of the decimal point for DECIMAL functions and two zeros for the cents in MONEY functions. For example, if C1 and C2 are INTEGER items with C1=2 and C2=5, and function C1/C2 is specified as DECIMAL, the results would be displayed as 0.000.

- For update commands, a stored function cannot contain Collect File items.

Examples

The following examples use the EMPLOYEE database. For each example, the function definition precedes the command invoking the function.

Example 1

This example shows invoking a simple INTEGER function in a retrieval command. By storing a function that calculates the age of an employee in the EMPLOYEE database, a label is provided in LIST output (even without title specifications) or for PRINT output when the NAME format option is in effect.

```
2010* EMPLOYEE AGE (INTEGER FUNCTION $
   ((*FTODAY*-BIRTHDAY)/365.25) $):
```

Invoking the function in the following command produces a report giving the EMPLOYEE NUMBER, LAST NAME, FORENAME, and EMPLOYEE AGE for those employees with employee numbers less than or equal to 1010.

```
LIST C1, C2, C3, *C2010* WH C1 <= 1010:
```

Example 2

This example invokes a simple function in an update command.

```
2011* TAKE HOME PAY (MONEY FUNCTION $
   (GROSS PAY - CURRENT DEDUCTIONS) $):
```

The following command, which invokes the above function, puts a value for NET PAY (C126) in the most recent (by specifying trace notation, *1*1*1) MONTHLY PAYROLL ACCOUNTING record. It is assumed that the record is there, but there is no current value for NET PAY.

```
AD C126 *1*1*1 = *C2011* WH C1 EXISTS:
```

INVOKING STORED PARAMETRIC FUNCTIONS

The only difference between a parametric function and a simple function is that a parametric function includes one or more parameters for which you supply values at the time the function is invoked.

You invoke a parametric function in the action-clause of retrieval or update commands by issuing the system separator, the function name or component number, followed by the appropriate values (enclosed in parentheses) for the parameters. For example,

```
*C2001(2400.00):
```

Format

```
* function (parameters)
```

function is the component name or number (including the C) of the stored function.

parameters are values for the numbered parameters that were defined in the function. The values you supply must be specified in the numeric order of the parameters, regardless of the order they appear in the function, and must be separated by commas. Parameters must be enclosed in parentheses.

Considerations

- You can use Collect File items as parameters for stored functions, including the (CF) indicator, but not in update commands.
- The same considerations that apply to stored simple functions apply to parametric functions.

Examples

The following examples use the EMPLOYEE database. For each example, the function definition precedes the command invoking the function. You use parameters in functions the same way that you use parameters in strings.

Example 1

This example defines a function to calculate the GROSS PAY (C124) of an employee who receives a commission.

```
2222* COMMISSION ( MONEY FUNCTION & (PAY RATE + *1* ) &):
```

In this case, the parameter (*1*) is for the amount of commission received by the employee.

You can then use the function in an update command, as shown below:

```
ADD C124 = *C2222 (3000.00) WH C121 EQ 05/31/91 AND C1 EQ 1031:
```

Because the software allows strings to invoke functions (and vice versa), the entire command can be stored in a string and the values for C121 and C1 can be made parameters.

```
2223*COMMISSION ADJUSTMENT (STRING $ ADD C124 = *C2222
   (*1*) WHERE C121 EQ *2* AND C1 EQ *3*:$):
```

You can then execute the above ADD command by invoking string C2223:

```
*C2223 (3000.00,05/31/91,1031)
```

Example 2

This example invokes a MONEY parametric function in an update command:

```
2020* BONUS GROSS PAY (MONEY FUNCTION $ (C111 + *1*) $):
```

Like invoking parametric strings, the system separator following the component name or number is replaced by the pair of parentheses containing the value supplied for the parameter.

```
AD C124 *1*1*1 = *C2020 (1500.00) WH C1 EQ 1006 AND C112 EQ
   MONTHLY/BONUS:
```

This is equivalent to issuing the following command:

```
AD C124 *1*1*1 = (C111 + 1500.00) WH C1 EQ 1006 AND C112 EQ
   MONTHLY/BONUS:
```

Note: if no data type had been specified for the function, the software would have converted the default DECIMAL value to MONEY since the results are being stored in a MONEY item. However, if the function were invoked in a retrieval command, the results of the function would be displayed as DECIMAL, that is, without a dollar sign.

System-Wide Commands

Introduction

Changing Processors

Designating Alternate User Files

Changing the Entry Terminator Word

Changing the System Separator

Changing the Where-Clause Delimiter

Changing the System Prompt

Echoing Commands

Inserting Comments

Monitoring Performance

Controlling Execution of Commands

Recording IF...THEN...ELSE Variables

Exiting a SYSTEM 2000 Session

Chapter 36

Introduction

System-wide commands are commands you can issue from any Self-Contained Facility processor.

For example, you can issue the EXIT command (a system-wide command) from any processor, but you can issue a LIST command only in the QUEST processor.

The following commands are system-wide commands:

COMMAND FILE IS
CONTROL
DATA FILE IS
DEFINE
DELIMITER IS
ECHO
ENTRY TERMINATOR IS
EXIT
IF...THEN...ELSE
LOOKUP IS
MESSAGE FILE IS
PROMPT IS
QUEST
REPORT FILE IS
SEPARATOR IS
SHOW
TIMING

Chapter 37

Changing Processors

USING PROCESSOR COMMANDS 37-1
Format 37-1
Considerations 37-1
Example 37-2

USING PROCESSOR LOOKUP 37-3
Format 37-3
Examples 37-4

USING PROCESSOR COMMANDS

The CONTROL, DEFINE, and QUEST (or ACCESS) commands call the CONTROL, DEFINE, and QUEST processors, so that you can issue commands specific to that processor. For example, if you want to issue QUEST language commands and you are using a different processor, you must first issue the QUEST command to call the QUEST processor.

To issue the DEFINE command, you must be using the master password.

You cannot issue CONTROL, DEFINE, or QUEST commands from the REPORT or QUEUE processors.

Format

```
CONTROL:
DEFINE:
QUEST:  (or ACCESS:)
```

Considerations

- Some SYSTEM 2000 commands automatically call a specific processor after execution. DATA BASE NAME IS, RESTORE, MAP, and GENERATE call the QUEST processor. NEW DATA BASE IS calls the DEFINE processor.

- To call the REPORT or QUEUE processors, you must be using the QUEST processor.

Example

```
USER, ABLE:
DBN IS EMPLOYEE:        (automatically calls the QUEST processor)
     .
     . (QUEST language commands)
     .
CONTROL:                (calls the CONTROL processor)
     .
     . (CONTROL language commands)
     .
QUEST:                  (calls the QUEST processor)
     .
     . (QUEST language commands)
     .
EXIT:                   (closes the database and ends the session)
```

USING PROCESSOR LOOKUP

The LOOKUP command gives you some flexibility in changing from one processor to another. If the processor lookup feature is active and you issue a command that is not valid for the current processor, the software will try to switch you to the appropriate processor.

Format

```
LOOKUP [IS] |OFF  :
            |ON
```

The LOOKUP IS command is a system-wide command, which can be be issued any time during an SCF session. ON activates the processor lookup feature. OFF (the default) disables the lookup.

Processor lookup switches to the appropriate processor if the first keyword in a command is unique to one processor. For example, if you issue SAVE from the QUEST processor, you will be given the CONTROL processor. However, if you issue PRINT SIZE (a CONTROL processor command) from the QUEST processor, you will remain in the QUEST processor because PRINT is a valid keyword for that processor.

Processor lookup is available only while you are using the QUEST or CONTROL processor. The DEFINE, REPORT, and QUEUE processors accumulate commands over the course of a session, so it would not be appropriate to lose accumulated (but unexecuted) commands because of a change in processors. Also, you must explicitly invoke the CONTROL processor in order to issue the RELEASE command. This prevents your releasing a database inadvertently from the QUEST processor.

A SYSTEM 2000 execution parameter (OPT045=YES/NO) is available to enable or disable the processor lookup feature at your site (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*). The default setting at installation time is OPT045=NO, which disables the lookup.

If the prompt is the standard three dashes and the software switches processors, you will receive the following message:

```
-899- USER NOW HAS -xxxxxxx- PROCESSOR -
```

where *xxxxxxx* is the name of the processor that was invoked.

If PROMPT IS PROCESSOR is in effect, you simply receive the new processor name as a prompt. For details about changing the prompt, see Chapter 42, "Changing the System Prompt: PROMPT IS."

If the software cannot switch processors, an appropriate message appears.

Examples

Example 1

This example shows a DESCRIBE command issued from the CONTROL processor when the system prompt is three dashes and processor lookup is active with LOOKUP IS ON.

```
 ---                                                    (default prompt)
control:                                                (input)
 CONTROL:                                               (echo)
 ---                                                    (default prompt)
print size:                                             (input)
 PRINT SIZE:                                            (echo)
                    EMPLOYEE
              PAGES       PAGES          PAGE
             IN USE      UNUSED          SIZE          (output)
                .           .             .
                .           .             .
                .           .             .
 ---                                                    (default prompt)
describe c1:                                            (input)
 DESCRIBE C1:                                           (echo)
  -899- USER NOW HAS -QUEST-  PROCESSOR -
      1*  ELEMENT-1 (CHAR X(6))                         (output)
 ---                                                    (default prompt)
```

Example 2

This example shows a DESCRIBE command issued from the CONTROL processor when the system prompt is the processor name and processor lookup is active with LOOKUP IS ON.

```
control:                                                (input)
 CONTROL:                                               (echo)
 CONTROL>                                               (processor prompt)
print size:                                             (input)
 PRINT SIZE:                                            (echo)
                    EMPLOYEE
              PAGES       PAGES          PAGE
             IN USE      UNUSED          SIZE          (output)
                .           .             .
                .           .             .
                .           .             .
 CONTROL>                                               (processor prompt)
describe c1:                                            (input)
 DESCRIBE C1:                                           (echo)
      1*  ELEMENT-1 (CHAR X(6))                         (output)
 QUEST>                                                 (processor prompt)
```

Chapter 38

Allocating and Changing the User Files

INTRODUCTION TO USER FILES 38-2

DYNAMIC ALLOCATION OF USER FILES 38-3

JCL FOR ALTERNATE USER FILES 38-4

LOCAL OPTION FOR CMS AND TSO 38-4

CHANGING COMMAND FILES: COMMAND FILE IS 38-5
- *Format 38-5*
- *Considerations 38-6*
- *Using the LOCAL Option 38-6*
- *Example 38-7*

CHANGING DATA FILES: DATA FILE IS 38-8
- *Format 38-8*
- *Considerations 38-9*
- *Using the LOCAL Option 38-9*
- *Examples 38-10*

CHANGING MESSAGE FILES: MESSAGE FILE IS 38-13
- *Format 38-13*
- *Considerations 38-14*
- *Using the LOCAL Option 38-14*
  - *TSO environment 38-15*
  - *CMS environment 38-15*
- *Examples 38-15*

CHANGING REPORT FILES: REPORT FILE IS 38-17
- *Format 38-17*
- *Considerations 38-18*
- *Using the LOCAL Option 38-19*
  - *TSO environment 38-19*
  - *CMS environment 38-19*
- *Examples 38-20*

EXAMPLES OF USING ALTERNATE USER FILES 38-21

INTRODUCTION TO USER FILES

Four user files are available for communication between you and SYSTEM 2000 software:

- The Command File is an input file from which the software reads input commands.
- The Data File is an input file from which the software reads values used in executing the LOAD command.
- The Report File is an output file on which the software writes the results of retrieval commands.
- The Message File is an output file on which the software writes error messages and informative messages.

S2KCOMD is the DDname for the Command File and the Data File. These files are initially set to INPUT, which is your terminal.

S2KMSG is the DDname for the Message File and the Report File. These files are initially set to OUTPUT, which is your terminal or a printer for batch jobs.

If you want to use an alternate file, you can change the user file name. You must designate the alternate file with an operating system statement and assign it an alternate DDname:

- In MVS batch, use a DD statement.
- In TSO, use an ALLOCATE statement.
- In CMS, use a FILEDEF statement.

The system-wide commands to designate alternate user files are

COMMAND FILE IS	designates an alternate Command File.
DATA FILE IS	designates an alternate Data File.
MESSAGE FILE IS	designates an alternate Message File.
REPORT FILE IS	designates an alternate Report File.

Dynamic allocation of S2KCOMD and S2KMSG is discussed next.

DYNAMIC ALLOCATION OF USER FILES

With Version 12 and later releases, you can let SYSTEM 2000 software allocate the user files dynamically. The following rules apply for dynamic allocation of S2KCOMD and S2KMSG.

1. SYSTEM 2000 software checks to see whether S2KCOMD or S2KMSG is already allocated in your JCL or CLIST. If so, the software uses that file; no dynamic allocation is performed.

2. If either or both files are not allocated, SYSTEM 2000 software allocates the file(s) dynamically according to the following rules:

   - S2KCOMD is set to the terminal for TSO, that is, DA(*). No allocation is done for non-TSO jobs, and the OPEN FAILED FOR S2KCOMD DATA SET message appears. You must include an S2KCOMD DD statement in batch jobs.

   - S2KMSG is set to the terminal for TSO, that is, DA(*). For non-TSO (batch or PLEX jobs), S2KMSG is set to SYSOUT=A.

If the software allocates a file dynamically, it also deallocates the file dynamically when the SYSTEM 2000 session ends.

JCL FOR ALTERNATE USER FILES

Alternate user files must be specified with JCL. In a single-user job, alternate user files are specified in your JCL. For Multi-User, alternate user files are specified with the Multi-User main task JCL for interactive processing except for local files in the dependent region. For a batch Multi-User job, alternate user files are specified in the dependent job step.

The file name used in a SYSTEM 2000 alternate user file command must have a corresponding JCL statement to assign the DDname. The format for that statement is

//DDname DD *file-assignment-and-disposition*

For example,

```
//ALTCOMND DD DSN=COMMANDS,DISP=OLD
//ALTREPRT DD SYSOUT=A
```

Then the SYSTEM 2000 commands specifying the alternate files would be

```
COMMAND FILE IS ALTCOMND:
REPORT FILE IS ALTREPRT:
```

To prevent unusual termination of a job, the last command in an alternate Command File should be one of the following:

```
COMMAND FILE IS INPUT:
COMMAND FILE IS another-DDname:
EXIT:
```

LOCAL OPTION FOR CMS AND TSO

For CMS or TSO TP users, a LOCAL option is available for the user files. The LOCAL option specifies a local alternate file in the dependent region.

- For CMS, issue the FILEDEF command to assign a name to a CMS local user file. For example, to assign the name MYNAME as a local file name for the Report File, specify

  ```
  FILEDEF ANYNAME DISK MYNAME DATA B
  ```

  The command for specifying the local Report File is

  ```
  LOCAL REPORT FILE IS ANYNAME:
  ```

  The Report File is

  ```
  MYNAME DATA B
  ```

  If the FILEDEF command had not been used, the Report File would be

  ```
  FILE ANYNAME A
  ```

- For TSO, the file must be allocated in the local TSO session and must be referenced using the DDname specified in the ALLOCATE statement. For example,

```
ATTR XYZ BLKSIZE(800) LRECL(80) RECFM(F) DSORG(PS)

ALLOC DS(MYFILE.DATA) DD(ANYNAME) SHR SPACE(1) TRACKS
USING(XYZ) NEW CATALOG
```

  The command for specifying the local Report File MYFILE is

```
LOCAL REPORT FILE IS ANYNAME:
```

CHANGING COMMAND FILES: COMMAND FILE IS

A Command File contains input commands to be read by the software. At the beginning of a session, the software sets the Command File to INPUT, which usually is your terminal or some other input device.

You can issue COMMAND FILE IS to direct the software to immediately start reading commands from a different Command File. To create an alternate Command File, use a text editor to write the commands, then store those commands in a data set.

Before you switch to a different Command File, you must first designate the alternate file with an operating system statement and assign it a DDname. See **JCL for alternate user files** on page 38-4.

Format

```
[LOCAL] COMMAND FILE IS  |INPUT   :
                         |fileref
```

LOCAL directs the software to look for the Command File in your TSO region or CMS machine.

fileref is the DDname of your alternate Command File. Do not use reserved file names. See Appendix E for a list of reserved file names.

After the COMMAND FILE IS command completes successfully, the Command File is automatically set to the beginning of the file.

Only one Command File can be open at a time. However, you can switch the Command File at any time and as often as needed in a session.

You cannot set the Command File and the Data File to the same physical file simultaneously. To use both (for instance, as in loading) one must be an alternate file. Also, input files cannot be the same as output files; therefore, the Command File cannot be set to the Message File or Report File.

The last command on an alternate Command File must be either EXIT to exit the session, COMMAND FILE IS INPUT to return to the standard input file, or COMMAND FILE IS *fileref*, telling the software to begin reading another alternate Command File. If the software does not find one of these commands, it automatically exits the session.

The COMMAND FILE IS command must be either the only command or the last command on an input line. The structure of the following example with multiple commands is acceptable to the software. However, it will not work correctly because the Command File (DEF) is executed immediately, and the rest of the commands on the line will be bypassed.

```
COMMAND FILE IS DEF:  MAP:  DATA FILE IS A: LOAD:
PRINT ENTRY:
```

The software opens file DEF and immediately reads the next record from DEF. In this example, DEF contains a database definition, but the MAP and LOAD commands will not be processed. The definition and data are not stored in the database, and PRINT ENTRY finds no data.

Considerations

An alternate file is usually a tape or a disk that has been allocated using a JCL DD statement or the TSO ALLOCATE statement. It has also been assigned a valid DDname.

The default logical record length and block size for the Command File is 80. However, you can specify an alternate record length of 1 to 250 characters. If you specify an alternate record length, you must also specify an alternate block size. The block size must be a multiple of the logical record length and must be in the range of 1 to 32,767 characters. The Command File must contain fixed-length records.

A DDname for each alternate Command File must be assigned, unless the file name is INPUT or the LOCAL option (for TSO and CMS) is specified. Also, to avoid conflict, alternate file names must be unique in a Multi-User environment.

If the Command File is a file with a standard label, do not specify its attributes. However, if the Command File is a nonlabeled file, the DD statement or ALLOCATE statement for the file must include LABEL=(,NL) and a DCB parameter with LRECL, BLKSIZE, and RECFM.

Using the LOCAL Option

If you are a CMS or TSO TP user, you can use the LOCAL option to specify a local alternate Command File, that is, a file not residing in the Multi-User region. The LOCAL COMMAND FILE IS command must be the last or only command in a TP segment. If not, the segment ends after the LOCAL COMMAND FILE IS command. Any commands left in that segment are lost.

A local alternate file must be preallocated, and it has the same restrictions as files used in a batch job. The LOCAL option is available only with the SYS2KTPI interface to the Multi-User software.

Switching to a local alternate Command File functions differently than switching to an alternate Command File in the Multi-User region. Issuing a command to switch the Command File in the Multi-User region causes any remaining commands in the old file to be lost. However, the segment does not end until you issue a command to switch back to the default or to a local alternate file.

If a local Command File is being read, the 1200-character limit for a segment does not apply. The entire file constitutes a segment.

Example

```
COMMAND FILE IS MYCOM:
```

The JCL for MYCOM is

```
//MYCOM DD DSN= data-set-name, DISP= |OLD
                                   |SHR
```

To return to the standard input file, the last command on MYCOM must be as follows:

```
COMMAND FILE IS INPUT:
```

CHANGING DATA FILES: DATA FILE IS

At the beginning of a session, the software sets the Data File to INPUT, which usually is your terminal or some other input device. A Data File contains a loader stream or value streams to be used for loading.

You can issue DATA FILE IS to tell the software to start reading data from a different Data File. To create an alternate Data File, use a text editor to write the data, then store them in a data set. You can also use the UNLOAD command to write data directly on the data set.

Before you switch to a different Data File, you must first designate the alternate file with an operating system statement and assign it a DDname. See **JCL for alternate user files** on page 38-4.

Format

```
[LOCAL] DATA FILE IS  |INPUT   :
                      |fileref
```

LOCAL directs the software to look for the Data File in your TSO region or CMS machine.

fileref is the DDname of your alternate Data File. Do not use reserved file names. See Appendix E for a list of reserved file names.

Only one Data File can be open at a time. However, you can switch the Data File at any time and as often as needed.

You cannot set the Command File and the Data File to the same physical file simultaneously. To use both files (for instance, as in loading) one must be an alternate file. Also, input files cannot be the same as output files; therefore, the Data File cannot be set to the Message File or the Report File.

The Data File is automatically set to the beginning of the file when the DATA FILE IS command completes successfully. For example:

```
1)  REPORT FILE IS A:
    UNLOAD:
        .
        .
        .
2)  REPORT FILE IS B:          (sets A to the beginning)
    UNLOAD:
        .
        .
        .
3)  REPORT FILE IS OUTPUT:     (sets B to the beginning)
        .
        .
        .
```

```
4)  DATA FILE IS B:  (uses output from 2)
          .
          .
          .
5)  DATA FILE IS A:  (sets B to the beginning)
                     (A is already set to the beginning.)
```

You cannot specify a file as both input and output concurrently. In the above example, the third command resets the Report File before the fifth command specifies the input Data File.

Considerations

An alternate file is usually a tape or a disk that has been allocated using a JCL DD statement or a TSO ALLOCATE statement. It has also been assigned a valid DDname.

The default logical record length and block size for the Data File is 80. However, you can specify an alternate record length of 1 to 250 characters. If you specify an alternate record length, you must also specify an alternate block size. The block size must be a multiple of the logical record length and must be in the range of 1 to 32,767 characters. The Data File must contain fixed-length records.

A DDname for each alternate Data File must be assigned, unless the file name is INPUT or the LOCAL option (for TSO and CMS) is specified. Also, to avoid conflict, alternate file names must be unique in a Multi-User environment.

If the Data File is a file with a standard label, do not specify its attributes. However, if the Data File is a nonlabeled file, the DD statement or the ALLOCATE statement for the file must include LABEL=(,NL) and a DCB parameter with LRECL, BLKSIZE, and RECFM.

Using the LOCAL Option

If you are a CMS or TSO TP user, you can use the LOCAL option to specify a local alternate Data File, that is, an alternate file not residing in the Multi-User region. The LOCAL DATA FILE IS command must be the last or only command in a TP segment. If not, the segment ends after the LOCAL DATA FILE IS command. Any commands left in that segment are lost.

A local alternate file must be preallocated, and it has the same restrictions as files used in a batch job. The LOCAL option is available only with the SYS2KTPI interface to the Multi-User software.

Switching to a local alternate Data File functions differently than switching to an alternate Data File in the Multi-User region. Issuing a command to switch the Data File in the Multi-User region causes any remaining commands in the old file to be lost. However, the segment does not end until you issue a command to switch back to the default or to a local alternate file.

If a local Data File is being read, the 1200-character limit for a segment does not apply. The entire file constitutes a segment.

Examples

Example 1

In this example, the LOAD command reads data from the alternate Data File MYDATA.

```
DATA FILE IS MYDATA:
LOAD:
```

The JCL for MYDATA is

```
//MYDATA DD DSN= data-set-name,DISP = |OLD
                                     |SHR
```

Example 2

If a Data File is composed of input lines in the input stream, you can use the following procedure:

```
DATA FILE IS INLINE:
```

The JCL contains

```
//INLINE  DD *
  (statements to be used as the Data File)
/*
```

Example 3

The Data File and the Command File cannot be the same file. However, if you are an interactive user in a single-user environment, you can circumvent this restriction when you attempt to load data from a terminal, as shown in this example.

An alternate Command File (on disk) can be set up containing a record as follows:

```
LOAD:  COMMAND FILE IS INPUT:
```

Assume this alternate file is allocated in a single-user environment with the DDname of AAAA. You can enter the following command to tell the software to read its next command from Command File AAAA. For example,

```
COMMAND FILE IS AAAA:
```

Notice that as soon as the current Command File ceases to be the default (INPUT), the software reads INPUT as the Data File. This is exactly what occurs when the software reads the first command on file AAAA. For example:

```
LOAD:
```

The LOAD command causes the terminal to be prompted for input; the software expects a standard loader stream to be entered. At this point, the current Command File is AAAA and the Data File is INPUT, that is, your terminal. This remains true until the software finds the logical end of the loader stream, which is two consecutive system separators (perhaps with blanks between them), for example, ** or *␣␣*.

When the two separators are found, the software completes the loading process and enters the values into the database tables. When the load process is complete, the software returns to the current Command File (AAAA) for the next command and finds the following command:

```
COMMAND FILE IS INPUT:
```

This command redirects the software to the original input file INPUT (terminal) for subsequent commands. Because the Data File is still INPUT, any attempt to enter a LOAD command again would cause an error message.

Example 3 (continued)

The following series of commands shows the case just described.

```
LOAD:  COMMAND FILE IS INPUT:  (contents of file AAAA)
   .
   .     (SYSTEM 2000 single-user interactive session)
   .
COMMAND FILE IS AAAA:
LOAD:  COMMAND FILE IS INPUT:  (parsed through LOAD:)
11:37:17 - BEGIN LOADING -
1*A*END*
1*B*END*
  .
  .
  .
END*
*                              (end of loader stream)
11:37:59 - SCAN COMPLETED -
  .
  .
  .
```

(The software reads COMMAND FILE IS INPUT: from file AAAA, then goes to the terminal for subsequent commands.)

```
PRINT ENTRY:                   (from user terminal)
  .
  .
  .
```

CHANGING MESSAGE FILES: MESSAGE FILE IS

At the beginning of a session, the software sets the Message File to OUTPUT, which normally is your terminal or a printer. A Message File contains messages from the software, which include error diagnostics, informative messages, warnings, abnormal job termination codes, and diagnostics. For example, the Message File contains SYSTEM 2000 Error Codes and STAE snapshot output. The Message File is also used for some messages during the execution of PLEX programs. In addition, if the ECHO ON command is in effect, the software writes echoes of input commands to the Message File.

You can issue MESSAGE FILE IS to tell the software to write messages and echoes of commands to a different Message File.

Before you switch to a different Message File, you must first designate the alternate file with an operating system statement and assign it a DDname. See **JCL for alternate user files** on page 38-4.

Format

```
[LOCAL] MESSAGE FILE IS  |OUTPUT   :
                         |fileref
```

LOCAL directs the software to look for the Message File in your TSO region or CMS machine.

fileref is the DDname of your alternate Message File. Do not use reserved file names. See Appendix E for a list of reserved file names.

Ordinarily, the Message File and the Report File are set to the same file. Setting the output files to the same file causes command echoes and retrieval results to be interleaved on the output device. That is, the echo of each command is immediately followed by its retrieval output and any pertinent messages.

When you switch to an alternate Message File, the software writes echoes and messages to the new file sequentially, apart from associated retrieval output. Switching to an alternate Message File is useful, for example, when producing labels, statements, or reports that require clean copy with no extra information. After a session, you can access alternate output file(s) by using the operating system.

Only one Message File can be open at a time. However, Message Files can be switched as often as necessary in a session.

The Message File is automatically set to the beginning of the file when the MESSAGE FILE IS command completes successfully. For example:

```
MESSAGE FILE IS A:
     .
     .
     .
MESSAGE FILE IS B:  (sets A to the beginning)
     .
     .
     .
```

The Message File cannot be the same file as the Command File or Data File.

The MESSAGE FILE IS command should be the only command on an input line. If other commands appear on the line, the echoes are sent to the current Message File.

Considerations

An alternate file is usually a tape or a disk that has been allocated using a JCL DD statement or a TSO ALLOCATE statement. Also, it has been assigned a valid DDname.

The default logical record length and block size for the Message File is 80. However, you can specify an alternate record length of 1 to 140 characters. If you specify an alternate record length, you must also specify an alternate block size. The block size must be a multiple of the logical record length and must be in the range of 1 to 32,767 characters. The Message File must contain fixed-length records.

If column 1 is to be interpreted as a carriage control, FBA must be specified in the RECFM subparameter of the DCB parameter in the DD statement or ALLOCATE statement that defines the file.

A DDname for each alternate Message File must be assigned, unless the name is OUTPUT or the LOCAL option (for TSO and CMS) is specified. Also, to avoid conflict, alternate file names must be unique in a Multi-User environment.

Using the LOCAL Option

The LOCAL option allows CMS or TSO TP users to specify a local alternate Message File (not in the Multi-User region).

For output with a destination in the dependent address space or machine, the software builds 4K output blocks and passes them to the interface when the block is full. Output to be displayed at the terminal is displayed from beginning to end with no browsing facility. However, the output is not displayed until all commands in a segment have been processed. For example, if you specify a local alternate Command File, the output is not displayed until the entire Command File has been processed. The output consists of all output generated by the Command File.

All terminal output is stored on the intermediate output file in 4K blocks until the processing for the current segment is complete. This output file is a QSAM file that must be allocated in the dependent address space or machine. The interface attempts to open the file during initialization. If the file does not exist, the software issues a nonfatal warning message.

The software does not attempt to write to the intermediate output file until there is more than one 4K block of output destined for the terminal. Then, if the file does not exist, the

remainder of the output for the segment is purged. At the end of the segment, the software displays one block of output and a second warning message appears indicating that output was purged.

A local alternate file must be preallocated, and it has the same restrictions as files used in a batch job. The LOCAL option is available only with the SYS2KTPI interface to the Multi-User software.

TSO environment If a B37 abend occurs on the intermediate S2KOUTP file while the interface is attempting to write to the file, the abend is trapped by an ESTAE routine. The remaining commands in the segment are processed, but all remaining output is truncated. At the end of the segment, the terminal output that was written to the file is presented, followed by an error message explaining that the output was truncated. A B37 abend for any local alternate output file is also trapped by the ESTAE routine, and the session ends.

The S2KOUTP file will be allocated dynamically if you do not allocate it with JCL or a CLIST. For more information about the intermediate S2KOUTP file, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

CMS environment An out-of-space condition for the intermediate file causes an I/O error. All I/O errors on the intermediate file or on any local alternate files are trapped by a SYNAD exit, and the job is ended by the software.

The existing commands for designating alternate input and output files function the same way as before. That is, if you issue a command to switch an input or output file to a file other than a local file, that file must exist in the Multi-User region. Because each input and output file can be specified separately, any combination of files in the dependent region and the Multi-User region (in addition to terminal input and output) is available. For example, the Command File could be a QSAM file in TSO with Message File output being written to the terminal and Report File output being written to a file in the Multi-User region.

Examples

Example 1

This example shows setting the Message File to an alternate disk file called MYMSG.

```
MESSAGE FILE IS MYMSG:
```

The JCL for MYMSG is

```
//MYMSG DD DSN= data-set-name,DISP= |NEW ,
                                   |MOD
                                   |OLD

  DCB=(BLKSIZE= desired-block-size,
  LRECL= desired-logical-record-length,
  RECFM=FBA)
```

Note that the record format is fixed blocked. Column 1 is interpreted as a carriage control character. If A is omitted, column 1 is treated as data.

Example 2

This example shows the Message File assigned to the printer, using S2KMSG as the file name.

```
MESSAGE FILE IS S2KMSG:
```

The JCL for S2KMSG is

```
//S2KMSG DD SYSOUT=A, DCB= ... (as shown in Example 1)
```

CHANGING REPORT FILES: REPORT FILE IS

At the beginning of a session, the software sets the Report File to OUTPUT, which is usually your terminal or a printer. The Report File contains results from retrieval commands from the QUEST, QUEUE, and REPORT processors.

You can issue the REPORT FILE IS command to have the software immediately begin writing retrieval output to a different Report File.

Before switching to a different Report File, you must first designate the alternate file with an operating system statement and assign it a DDname. See **JCL for alternate user files** on page 38-4.

Format

```
[LOCAL] REPORT FILE IS  |OUTPUT  :
                        |fileref
```

LOCAL directs the software to look for the Report File in your TSO region or CMS machine.

fileref is the DDname of your alternate Report File. Do not use reserved file names. See Appendix E for a list of reserved file names.

Ordinarily, the Report File and the Message File are set to the same file. Setting the output files to the same file causes retrieval results and echoes of commands to be interleaved on the output device. That is, the echo of each command is immediately followed by its retrieval output and any pertinent messages.

When you switch to an alternate Report File, the software writes the results of retrievals to the new file sequentially, apart from command echoes and messages. Switching to an alternate Report File is useful, for example, when sending retrieval results to different files for printing on special forms (for example, two-part paper or label paper). Also, by using an alternate Report File, different reports to be distributed to different groups can be separated for special off-line printing. You can also use an alternate Report File to hold a long, time-consuming printout so that it can be printed off-line at some time other than during prime production hours. After a session, you can access alternate output file(s) by using the operating system.

Only one Report File can be open at a time. However, Report Files can be switched as often as necessary in a session.

The Report File is automatically set to the beginning of the file when the REPORT FILE IS command completes successfully. For example:

```
1)  REPORT FILE IS A:
        .
        .
        .
2)  REPORT FILE IS B:        (sets A to the beginning)
        .
        .
        .
3)  REPORT FILE IS OUTPUT:   (sets B to the beginning)
        .
        .
        .
4)  DATA FILE IS B           (uses output from 2)
        .
        .
        .
5)  DATA FILE IS A           (sets B to the beginning)
                             (A is already set to the beginning.)
```

You cannot specify a file as both input and output simultaneously. In the example above, the third command resets the Report File before the fourth command specifies the input Data File.

The REPORT FILE IS command should be the only command on an input line. If other commands appear on the line, their results are sent to the current Report File.

If you attempt to switch to an alternate file and the file cannot be opened, the software switches to the default OUTPUT file.

Considerations

An alternate file is usually a tape or a disk that has been allocated using a JCL DD statement or a TSO ALLOCATE statement. Also, it has been assigned a valid DDname.

The default logical record length and block size for the Report File is 80. You can, however, specify an alternate record length of 1 to 140 characters. If you specify an alternate record length, you must also specify an alternate block size. The block size must be a multiple of the logical record length and must be in the range of 1 to 32,767 characters. The Report File must contain fixed-length records.

If column 1 is to be interpreted as a carriage control, FBA must be specified in the RECFM subparameter of the DCB parameter in the DD statement or ALLOCATE statement that defines the file.

A DDname for an alternate Report File must be assigned, unless the file name is OUTPUT or the LOCAL option (for TSO and CMS) is specified. Also, to avoid conflict, alternate file names must be unique in a Multi-User environment.

Using the LOCAL Option

The LOCAL option allows CMS and TSO TP users to specify a local alternate Report File (not in the Multi-User region).

For output with a destination in the dependent address space or machine, the software builds 4K output blocks and passes them to the interface when the block is full. Output to be displayed at the terminal is displayed from beginning to end with no browsing facility. However, the output is not displayed until all commands in a segment have been processed. For example, if you specify a local alternate Command File, the output is not displayed until the entire Command File has been processed. The output consists of all output generated by the Command File.

All terminal output is stored on the intermediate output file in 4K blocks until the processing for the current segment is complete. This output file is a QSAM file that must be allocated in the dependent address space or machine. The interface attempts to open the file during initialization. If the file does not exist, the software issues a nonfatal warning message.

No attempt to write to the intermediate output file is made until there is more than one 4K block of output destined for the terminal. Then, if the file does not exist, the remainder of the output for the segment is purged. At the end of the segment, the software displays one block of output and a second warning message indicating that output was purged.

A local alternate file must be preallocated, and it has the same restrictions as files used in a batch job. The LOCAL option is available only with the SYS2KTPI interface to the Multi-User software.

TSO environment If a B37 abend occurs on the intermediate S2KOUTP file while the interface is attempting to write to the file, the abend is trapped by an ESTAE routine. The software processes the remaining commands in the segment but truncates all remaining output. At the end of the segment, the terminal output that was written to the file is presented, followed by an error message explaining that the output was truncated. A B37 abend for any local alternate output file is also trapped by the ESTAE routine, and the session ends.

The S2KOUTP file will be allocated dynamically if you do not allocate it with JCL or a CLIST. For more information about the intermediate S2KOUTP file, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

CMS environment An out-of-space condition for the intermediate file causes an I/O error. All I/O errors on the intermediate file or on any local alternate files are trapped by a SYNAD exit, and the job is ended by the software.

The existing commands for designating alternate input and output files function the same way as before. That is, if you issue a command to switch an input or output file to a file other than a local file, that file must exist in the Multi-User region. Because each input and output file can be specified separately, any combination of files in the dependent region and the Multi-User region (in addition to terminal input and output) is available. For example, the Command File could be a QSAM file in TSO with Message File output being written to the terminal and Report File output being written to a file in the Multi-User region.

Examples

Example 1

This example shows switching to an alternate Report File.

```
REPORT FILE IS MYREPORT:
```

The JCL for MYREPORT is

```
//MYREPORT DD DSN= data-set-name,DISP= |NEW ,
                                      |MOD
                                      |OLD
  DCB=(BLKSIZE= desired-block-size,
  LRECL= desired-logical-record-length,
  RECFM=FBA)
```

Note that the record format is fixed-block. Column 1 is interpreted as a carriage control character. If A is omitted, column 1 is treated as data.

Example 2

This example shows the Report File assigned to the printer.

```
REPORT FILE IS RPTOUT:
```

The JCL for RTPOUT is

```
//RPTOUT DD SYSOUT=A, DCB= ... (as shown in Example 1)
```

EXAMPLES OF USING ALTERNATE USER FILES

Example 1

In this example, the REPORT FILE IS command directs output from retrieval commands to alternate Report Files with different DDnames. (You can assign each Report File to a different data set to be stored on disk, viewed at a terminal, or printed. The session includes DEFINE language commands to define a string.)

```
// EXEC S2K
   .
   .
   .
//MARKETIN DD dataset-definition 1
//INFOSYST DD dataset-definition 2
//ADMNFIN  DD dataset-definition 3
USER, DEMO:
DBN IS EMPLOYEE:
DEFINE: 4
1001*MONTHLY REPORT (STRING $LIST C1, C2, C3, C105, OB C2
   WHERE C102 EQ *1* AT 1:$): 5
MAP:
STOP IF ANY COMMAND IS REJECTED: 6
REPORT FILE IS MARKETIN: 7
*MONTHLY REPORT (MARKETING): 8
REPORT FILE IS INFOSYST: 9
*MONTHLY REPORT (INFORMATION SYSTEMS): 10
REPORT FILE IS ADMNFIN: 11
*MONTHLY REPORT (/ADMINISTRATION & FINANCE/): 12
EXIT:
/*
```

Example 1 (continued)

1 This DD statement defines a file with DDname MARKETIN.

2 This DD statement defines a file with DDname INFOSYST.

3 This DD statement defines a file with DDname ADMNFIN.

4 The system-wide DEFINE command calls the DEFINE processor.

5 The string MONTHLY REPORT is defined. It includes a LIST command and a parameter for the value of C102 to be supplied when the string is invoked.

6 The STOP IF command guarantees that the session will stop if there is any error.

7 File MARKETIN is specified as the Report File.

8 String MONTHLY REPORT is invoked to execute a LIST command with the value of MARKETING for C102. The output of this LIST command is written on file MARKETIN.

9 File INFOSYST is specified as the Report File.

10 String MONTHLY REPORT is invoked to execute a LIST command with the value of INFORMATION SYSTEMS for C102. The output of this LIST command is written on file INFOSYST.

11 File ADMNFIN is specified as the Report File.

12 String MONTHLY REPORT is invoked to execute a LIST command with the value of ADMINISTRATION & FINANCE for C102. The output of this LIST command is written on file ADMNFIN.

Example 2

In this example, the REPORT FILE IS, COMMAND FILE IS, and DATA FILE IS commands create a test database called TESTRUN. The JCL shown includes the database files that were allocated for the TESTRUN database prior to the execution of this procedure. One standard technique for defining a database is to keep the definition in an alternate Command File, edit the file as needed during planning, and use the file to create a test database definition when appropriate. In this case, the schema for the EMPLOYEE database is stored in the data set S2K.PUB.EMPSCHEM.

```
// EXEC S2K
   .
   .
   .
//EMPSCHEM DD DSN=S2K.PUB.EMPSCHEM,DISP=OLD 1
//TEMPDATA DD SPACE=(CYL,(1,1)),UNIT=WK30, 2
// DCB=(BLKSIZE=1600,LRECL=80,RECFM=FB)
//SYSIN    DD *
USER, DEMO:
DBN IS EMPLOYEE:
REPORT FILE IS TEMPDATA: 3
UNLOAD C0 WHERE C102 EQ MARKETING AT 1: 4
REPORT FILE IS OUTPUT: 5
PRINT COUNT C0 WHERE SAME: 6
CONTROL: 7
NEW DATA BASE IS TESTRUN: 8
COMMAND FILE IS EMPSCHEM: 9
MAP: 10
DATA FILE IS TEMPDATA: 11
LOAD: 12
PRINT COUNT C0: 13
EXIT:
/*
```

Example 2 (continued)

1 This DD statement identifies the file where the schema for the EMPLOYEE database is kept. The DDname is EMPSCHEM. The first command stored in the file is

```
1* EMPLOYEE NUMBER (INTEGER 9999):
```

The last two commands stored in the file are

```
423* DATE ACCOMPLISHED (DATE IN 420):
COMMAND FILE IS INPUT:
```

The other commands define the other components of the EMPLOYEE database as shown in the foldout at the end of the manual.

2 This DD statement defines a temporary data set (file) with DDname TEMPDATA.

3 File TEMPDATA is identified as the Report File.

4 This UNLOAD command retrieves all entries for employees currently working in the Marketing Department. UNLOAD is used instead of LIST or PRINT, because the resulting output is in a format suitable for input into another database.

5 OUTPUT is specified as the Report File. This command serves two purposes: it directs output from the next retrieval command to the printer, and it also allows the TEMPDATA file to be used as a Data File. Remember that a file cannot be a Report File and a Data File at the same time.

6 This command requests the number of logical entries that were unloaded. Notice that the SAME operator is used.

7 The system-wide CONTROL command calls the CONTROL processor. The CONTROL processor is required to issue the CONTROL language command NEW DATA BASE IS.

8 This command establishes a new database and calls the DEFINE processor.

9 The EMPSCHEM file is specified as the Command File. This causes the software to read the commands stored in that file and use them to define the schema for the TESTRUN database. The last command in that file, COMMAND FILE IS INPUT, causes the software to return to the INPUT file in order to read the next command, in this case, the MAP command.

10 The DEFINE language MAP command executes the commands in file EMPSCHEM and maps the definition. After this command is executed, TESTRUN and EMPLOYEE have the same schema.

11 The TEMPDATA file is specified as the Data File. In step 3, data for all employees who are currently in the Marketing Department were unloaded to file TEMPDATA; here the data are loaded into the TESTRUN database.

12 The LOAD command loads the data in TEMPDATA into the TESTRUN database.

13 This retrieval command checks whether the number of entries loaded into TESTRUN agrees with the number of entries unloaded into file TEMPDATA. In other words, it should correspond to the number retrieved by step 6.

Chapter 39

Changing the Entry Terminator Word: ENTRY TERMINATOR IS

Format 39-1
Example 39-1

The entry terminator word signals end-of-data for a data tree in a loader stream or in an update command. The default entry terminator word is END.

The entry terminator word and the system separator form the entry terminator. The default system separator is the asterisk (*). Therefore, by default, the entry terminator is END*.

You can issue the ENTRY TERMINATOR IS command to change the entry terminator word. However, the only reason to change the entry terminator word is for readability. No conflict occurs if END is a value in a value stream.

Format

```
ENTRY TERMINATOR IS word :
```

word is a name to be used as the entry terminator word. The name can be from one to four alphanumeric characters with no blanks. If you use a special character, it must be only one character, and it cannot be the current system separator or command terminator. See Appendix C for a list of available special characters.

If the master password holder changes the entry terminator word, the word is changed for all users of that database. After the master password holder changes the word, any user opening that database must use the new word. However, if a user already has the database open before the master password holder changes the word, the database is not affected for that user until it is closed and reopened.

If a secondary password holder changes the entry terminator word, the word is changed only for that database in that session.

Example

```
ENTRY TERMINATOR IS FINI:

IT C120*1*1*1 EQ 121*06/30/91* FINI* WH C1 EXISTS:
```

Chapter 40

Changing the System Separator: SEPARATOR IS

Format 40-1
Uses of the system separator 40-2
Uses of the system separator for output 40-3
Checking for the system separator in UNLOAD output 40-3
Example 40-3

The system separator separates labels from values and also marks the end of an input value. The system separator delimits values during data entry, in retrieval and update commands, and in many output displays. The default is the asterisk.

You can issue the SEPARATOR IS command to change the system separator. However, if you need to use the asterisk within data, change the system separator to a character that does not conflict with other symbols being used. (For example, the colon cannot be used as the system separator, because it is the command terminator.)

Format

```
SEPARATOR IS symbol  :
```

symbol is a single special character used as the system separator. The system separator cannot be used in values, and it cannot be the current where-clause delimiter or command terminator. See Appendix C for a list of available special characters.

If the master password holder changes the system separator, the separator is changed for all users of that database. After the master password holder changes the system separator, any user opening that database must use the new separator. However, if a user already has the database open before the master password holder changes the system separator, the database is not affected for that user until it is closed and reopened.

If a secondary password holder changes the system separator, the system separator is changed only for that database in that session.

Uses of the system separator For input operations, the system separator needs to be used

- immediately following a component number when you are defining a component

```
1* STUDENTS NAME (KEY NAME):
```

- immediately following each component number and each value when you are specifying a value stream in an update command

```
1*SMITH*2*123-45-6789*3* ...
```

- on both sides of a stored simple function, either function name or component number

```
PRINT *C2000* WHERE ...
```

- on the left side of a stored parametric function

```
PRINT *FN1(C10):
```

- on both sides of a stored simple string, either string name or component number

```
*STRING10* WHERE C1 EXISTS:
```

- on the left side of a stored parametric string, either string name or component number

```
*STRING1(100)
```

- on both sides of a system string

```
*DATA*, *NOW*, *TODAY*
```

- immediately following the entry terminator word END, to form the entry terminator

```
END*
```

- to indicate end-of-data on the Data File

```
END**
```

- immediately following the component reference in the item-to-item comparison capability

```
WHERE C1* EQ C2*
```

- in trace notation within QUEST language update commands

```
REMOVE TREE ENTRY * 4:
```

- as a symbolic parameter delimiter within a stored parametric string or function definition.

```
... WH C1 EQ *1*:
```

Uses of the system separator for output For output, the separator occurs

- immediately after the stub in PRINT command output

```
1* 10465
2* SMITH
```

- in loader stream output from the UNLOAD command.

```
... 1* 10465*2*SMITH* ...
```

Checking for the system separator in UNLOAD output The software checks to see whether the current system separator occurs in any CHARACTER, TEXT, or UNDEFINED value being unloaded. If so, message -290- appears.

You can disable this checking with the OPT000 execution parameter (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*). You can also request a check for all characters that can be specified as the system separator.

Example

```
SEPARATOR IS $:
REMOVE TREE ENTRY $4:
```

Chapter 41

Changing the Where-Clause Delimiter: DELIMITER IS

Format 41-1
Multi-User Consideration 41-1
Examples 41-2

The where-clause delimiter encloses a value in a where-clause condition for QUEST commands, for the DEFINE language MAP command, and for the REPORT language GENERATE command. The where-clause delimiter allows you to qualify values containing keywords, such as AND, OR, and NOT, or values containing more than one consecutive blank.

The slash is the default, which is set at the beginning of each session.

You can issue the DELIMITER IS command to change the where-clause delimiter. If you change the delimiter during a session, the change is valid only for the duration of that session.

Format

```
DELIMITER IS  delimiter  :
```

delimiter is a single special character used as the where-clause delimiter. The delimiter cannot be used in values, and it cannot be the current system separator or command terminator. See Appendix C for a list of the special characters available.

Multi-User Consideration

Since the system separator is stored in the database for all users, if a user in a Multi-User environment changes the system separator, this change could cause a conflict. Users in a Multi-User environment should establish workable rules to avoid conflicts. Each user should establish a where-clause delimiter, making sure it avoids conflict with the system separator.

Examples

Example 1

This example shows a command stream containing a change in the where-clause delimiter.

```
DELIMITER IS $:
PRINT ENTRY WHERE C303 = $CHURCH AND SCHOOL$:
DITTO WHERE C303 GT $SCHOOL/UNIVERSITY$:
```

Example 2

Examples of delimiters and the values to be used in the qualification process are shown in this example. For the values below, suppose that C1 is type CHARACTER and C2 is type TEXT.

Delimited Values	Values Used
WHERE C1 GT /ABCbb/:	ABCbb
WHERE C1 EQ /bA/:	bA
WHERE C1 EQ /A/*/AA/:	A and AA
WHERE C1 EQ /A/*bbb/CbbAT/:	A and CbbAT
WHERE C2 EQbbbbA:	bbbA
WHERE C2 EQbbbb/A/:	A
WHERE C2 LEbbbb/bbA/:	bbA
WHERE C2 EQbbbb/bA/*bbb/bCAT/:	bA and bCAT

Chapter 42

Changing the System Prompt: PROMPT IS

Format 42-1
Considerations 42-1
Examples 42-2

The PROMPT IS command allows you to choose another prompt during an SCF session. Instead of the standard three dashes (---), the system prompt can be the name of the current processor. Or you can turn the prompt off.

Format

```
PROMPT [IS] |DASH      :
            |PROCESSOR
            |OFF
```

DASH (the default) activates the standard prompt of three dashes.

PROCESSOR specifies that the prompt is the name of the current processor, for example,

```
CONTROL>
DEFINE>
QUEST>
REPORT>
```

OFF suppresses the system prompt altogether.

Considerations

- The ECHO ON/OFF command still determines whether the prompt is issued, regardless of which prompt is specified.
- SYSTEM 2000 execution option OPT044=YES/NO is available to reset the prompt at your site (see v12psml.). The default setting at installation time is three dashes (OPT044=NO).

Abbreviations and synonyms:
DASH = DASHES, DASHS, ON, or the dash (-)
PROCESSOR = PROC

- If you have written programs that read SYSTEM 2000 output files and the default prompt at your site was changed to be the processor name, you can change the prompt in your jobs to the standard prompt (---) with the PROMPT IS command. This will make the SYSTEM 2000 output compatible with your programs.

- For Multi-User batch jobs, the prompt is always three dashes.

Examples

Here are some examples of this command:

```
PROMPT PROC:

PROMPT IS - :

PROMPT OFF:
```

Chapter 43

Echoing Commands: ECHO

Format 43-1
Echoes of Input Commands 43-1
Echoes of Strings 43-2
Consideration 43-2
Examples 43-3

The ECHO command writes comments or SYSTEM 2000 commands to the Report File and directs the software to write (echo) input commands to the Message File.

For batch jobs, the default is ECHO ON. For interactive sessions, the default is ECHO OFF. The software sets the defaults at the beginning of each session. If you change the ECHO status during a session, the change is valid only for the duration of the session.

Format

```
ECHO    |ON                                    :
        |OFF
        |delimiter  string  delimiter
```

delimiter is a single special character used as the ECHO string delimiter. The default is a slash. The delimiter cannot be the current system separator or command terminator. See Appendix C for a list of the special characters available.

string defines a string of characters not containing the character used for the delimiter. The maximum is 249 characters. You can issue as many ECHO string commands as needed.

Echoes of Input Commands

ECHO ON causes the software to write echoes of input commands to the Message File. Because the Message File and the Report File are often the same output file, the echoes are interleaved in sequence immediately before the output for each command.

The default for batch jobs is ECHO ON, which allows each command to be displayed before its output. If ECHO OFF is specified, only the output and any error messages are displayed.

The default for an interactive session is ECHO OFF, because the commands are usually visible when they are keyed in. However, you may want to issue the ECHO ON command to verify that the keyed-in commands were transmitted properly.

If the ECHO ON command is in effect when you issue the ECHO OFF command, ECHO OFF displays to show that the setting has changed. The ECHO OFF command is the last command echoed until you issue ECHO ON again. On the other hand, if ECHO OFF is in effect and you issue ECHO ON, ECHO ON does not appear on the Message File; echoes begin with the next command issued.

Echoes of Strings

If a string of characters (enclosed by delimiters) is specified in an ECHO command, all characters in the string are written directly to the Report File. Use this form of the command when you want to build a file of commands on an alternate Report File that can be used as a Command File later on.

The characters in the string are written as 80-character lines. For example, one ECHO command with a string could write up to three 80-character lines followed by a 9-character line.

The ECHO command with a string provides a convenient way to create a series of commands without your having to use a text editor outside the software.

For example, when you are using the DESCRIBE/DEFINE/ or DESCRIBE/COLLECT FILE/ command, the DESCRIBE output is written to the Report File in DEFINE language format. To make a complete set of DEFINE commands and specify an alternate Report File, issue ECHO commands to surround the DESCRIBE output, as shown in the following example:

```
REPORT FILE IS ABC:
ECHO/DEFINE:/:
DESCRIBE/DEFINE/:
ECHO/MAP:/:
ECHO/COMMAND FILE IS INPUT:/:
```

Another use of the ECHO command with a string is for commenting the results of commands, such as PRINT or LIST commands. The string can contain any set of alphanumeric characters as a comment.

If stored strings or functions are specified in an ECHO command with a string, they are not expanded until later, when they are read and processed as a Command File. The characters given in the string are written to the Report File exactly as found.

If a string is null (that is, ECHO //:), nothing is written to the Report File.

Consideration

If you issue the ECHO command within a stored simple or parametric string, the string is always written to the Report File regardless of the default.

Examples

Example 1

```
ECHO ON:
   .
   . (echoes of commands and results)
   .
ECHO OFF:
   .
   . (results only)
   .
```

Example 2

```
ECHO/MAP:COMMAND FILE IS INPUT:/:    (writes MAP:COMMAND FILE IS
                                      INPUT: on the Report File)

ECHO/THIS IS A COMMENT/:             (writes THIS IS A COMMENT on
                                      the Report File)
```

Chapter 44

Inserting Comments

Format 44-1
Considerations 44-1
Examples 44-2

You can insert comments after any CONTROL, DEFINE, QUEST, QUEUE, or REPORT language command. The comments are written to the Message File if the ECHO ON command is in effect. The comments can serve many purposes, such as explanations for job streams or archival purposes.

Format

delimiter comment delimiter

delimiter is a single special character used as the comment delimiter. The delimiter cannot be the current system separator or the command terminator. Also, the beginning and ending delimiter must be the same. See Appendix C for a list of available special characters.

comment defines a string of 1 to 250 characters. The string of characters cannot contain the delimiter.

The software writes comments to the Message File exactly as you enter them; that is, consecutive blanks are preserved. Consecutive blanks are counted as one space in the 250 character maximum, but single blanks are counted individually.

You can insert a comment anywhere after a command terminator but before the next command. A comment, like all Self-Contained Facility commands, can span input lines.

Considerations

- For a batch job, the default is ECHO ON. For an interactive session, the default is ECHO OFF.
- When the software fails to find the start of a valid command, it looks for a beginning delimiter. If it does not find one, the software issues an error message and scans to the next command terminator before attempting to resume processing. Therefore, if a command follows a comment with an invalid beginning delimiter, the software does not process the command.

  You can follow each comment with a colon so that the software will process commands following the comment even if the comment contains an invalid beginning delimiter.

- When the software fails to find an ending delimiter that is the same special character as the beginning delimiter, it scans up to 250 characters and issues an error message. Everything after the beginning delimiter is treated as a comment.

Examples

In the following examples, the Message File and the Report File are the same file.

Example 1

The following example shows DEFINE language commands with comments:

```
---
DEFINE:
---
1*EMPLOYEE NUMBER (INTEGER 9999): & Comment after command &
---
3*AA(CHAR X(20)):  //  /The preceding is a null comment/
---
```

Example 2

The following example shows QUEST commands that include comments using various rules. Note that the input line showing the command terminator is at the end of a comment.

```
---
IT C0*0 EQ 1*5*3*NNNN*5*6*A*END*:
 -342-       1 SELECTED RECORDS(S) -
---
# System separator (*) & command terminator (:) are OK inside a comment #
---
IT C0*0 EQ 1*7*3*QQQ*END*:
 -342-       1 SELECTED RECORD(S) -
---
Comment with no delimiters creates an error
C
 -100- SYNTAX ERROR -
---
 TALLY C3:  / The preceding TALLY C3 is lost via scan to : /
---
* The system separator cannot be a comment delimiter *  :
.....................................................C
 -808- UNRECOGNIZED COMPONENT LABEL -
---
PR COUNT C1: /PRINT command executed because : follows the bad comment/
CNT 1*  2
---
IT C0*0 EQ 1*8*3*PPP*END*:/Place comment anywhere after a command/
 -342-       1 SELECTED RECORD(S) -
---
: Colon cannot be the comment delimiter:
.....C
 -100- SYNTAX ERROR -
---
PRINT COUNT ENTRY:
CNT 0*  3
---
            /* This comment appears to have C-programming-style comment */
---
PR COUNT C6: /* delimiters, but an internal slash is not allowed */
CNT 6*  1
---
```

Example 3

The following example shows a comment that is too long. Note that consecutive blanks are not included in the 250 character maximum count, but single blanks are counted.

```
---
a                  This comment is a very long comment.
---
                    It contains more than 250 characters.
---
Comments are limited in size because they are collected in a 250-
---
character SYSTEM 2000 buffer.  Consecutive blanks within a comment are
---
removed before being counted in the maximum character count of 250.  a
.....................................................C
 -804- TOO MANY CHARACTERS IN VALUE OR NAME -
---
```

Example 4

The following example shows how to insert a comment that is longer than 250 characters by using delimiters on each comment line.

```
---
+ This comment is logically greater than 250 characters.      +
---
+ However, it is treated as several comments because each     +
---
+ line has its own set of delimiters.  The size of each       +
---
+ individual line does not violate the size restriction       +
---
+ established for processing each string.                     +
---
```

Example 5

The following example shows the output from a comment that is inserted on the same line as a command. The comment crosses a line boundary. The output for the TALLY command is displayed before the second line of the comment because the software scans the TALLY command to the command terminator, processes the command, and then scans and processes the rest of the line.

```
---
TALLY C1:  / There is no restriction about going over
*****************************************
 ITEM-         EMPLOYEE NUMBER
*****************************************
OCCURRENCES   VALUE
-----------------------------------------
       1          5
       1          7
       1          8
-----------------------------------------
       3 DISTINCT VALUES
-----------------------------------------
       3 TOTAL OCCURRENCES
-----------------------------------------
---
           the end-of-line boundary with a comment.
---
           Note the appearance of these lines. /
---
```

Example 6

You can also include commands that you can use in a job stream at a later time. For example, if you are testing a job stream, you might want to include commands that you will use for production runs. This example shows a command delimited as a comment. The command switches the Message File to an alternate file. After testing, the delimiters can be removed for a production run.

```
---
USER,DEMO:
---
 & MESSAGE FILE IS BCFCMT: &
---
NDB IS LIBRARY:
---
```

Chapter 45

Monitoring Performance: TIMING

Format 45-1
Example 45-2

The TIMING command reports the following statistics for each SYSTEM 2000 command issued while TIMING ON is in effect:

- each SYSTEM 2000 overlay that is called
- the number of I/O requests (read or write) to the database tables and scratch files
- cumulative I/O statistical tables for each session.

The statistics are written to the Message File. At the beginning of each session, timing is set to OFF.

Format

```
TIMING |OFF  :
       |ON
```

Five lines of information are written for each SYSTEM 2000 overlay that is called during the execution of a command.

The individual and cumulative total I/O statistics are summary in nature and do not include sort/merge I/O counts. More comprehensive and detailed I/O count and CPU utilization statistics can be obtained using the QAEXIT/QASTAT utility that is documented in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

Some installations may use a flat system rather than an overlay structure, so that no overlays need be loaded during execution. In this case, output from the TIMING ON command still shows the timing for each overlay at the time it would have been loaded.

Timing information is written to the Message File. The software writes five lines for each SYSTEM 2000 overlay that is called during command execution. The member names and general function of each overlay are documented in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

```
EXITING OVERLY      =  prim, sec
FILE1               =  n DVT= nEFT= n
MOT                 =  n HT= nDT= n
TBLIO               =  table-I/Os SCRIO= scratch-I/Os TIO= total-I/Os
TIMING FOR OVERLAY  =  prim, sec CPU-time
```

prim	=	primary overlay number
sec	=	secondary overlay number
n	=	number of I/O requests for individual database tables
table-I/Os	=	number of I/O requests to the database tables
scratch-I/Os	=	number of I/O requests to the scratch files or scratch pads
total-I/Os	=	total number of I/O requests (not including sort/merge I/O)
CPU-time	=	CPU time in hundredths of seconds

Example

To look at cumulative statistics for a queue or load session, turn timing on and off.

```
TIMING OFF.              (Turn off before the QUEUE command.)
    .
    .
    .
QUEUE.
    .
    .
    .
TIMING ON.               (Turn on before the TERMINATE command.)

TERMINATE.

TIMING OFF.              (Turn off after the QUEUE/TERMINATE session,
                         unless you want to gather more timing statistics.)
```

Chapter 46

Controlling Execution of Commands: IF...THEN...ELSE

Format 46-1
Considerations 46-4
Examples 46-5

The IF...THEN...ELSE commands allow you to conditionally execute any sequence of CONTROL, DEFINE, or QUEST commands. The IF criteria allow you to test the following system variables:

- the completion code (indicating whether the command completed successfully)
- the message number issued
- the number of data records selected
- the master password or any secondary password
- the database cycle number.

Additionally, you can use the SHOW command to write the values for these system variables (except the password) to the Message File for debugging purposes. See Chapter 47, "Recording IF...THEN...ELSE Variables: SHOW."

You cannot use IF...THEN...ELSE logic in the REPORT language or in the QUEUE language. Both of these languages have their own type of if-clause processing.

Format

```
IF  expression  THEN  commands ...
                [ELSE  commands ...]
                ENDIF  :
```

Abbreviations:
COMPLETE = C
DATA BASE = DB
ERROR = E
MASTER PASSWORD = MPW
MESSAGE = MSG
PASSWORD = PW
RECORDS = RECS
WARNING = W

expression

```
|condition
|NOT expression
|(expression)
|expression [AND expression]
|expression [OR expression]
```

condition

```
|COMPLETE
|WARNING [OCCURRED]
|ERROR [OCCURRED]
|MESSAGE [NUMBER] binary-operator message
|MESSAGE [NUMBER] ternary-operator message * message
|[SELECTED] RECORDS binary-operator integer
|[SELECTED] RECORDS ternary-operator integer * integer
|DATA BASE CYCLE binary-operator integer
|DATA BASE CYCLE ternary-operator integer * integer
|PASSWORD binary-operator password
|PASSWORD ternary-operator password * password
|MASTER PASSWORD
```

binary-operator LT, LE, NE, EQ, GT, or GE. See **Comparing an Item with a Value Using EQ, NE, GE, GT, LE, or LT** on page 24-9, **Comparing Two Items Using EQ, NE, GE, GT, LE, or LT** on page 24-11, or **Comparing an Item with a Collect File Item** on page 24-12.

ternary-operator EQ, NE, or SPANS. See **Comparing an Item with a Range of Values Using EQ, NE, or SPANS** on page 24-15.

message is a Self-Contained Facility message number.

integer is a positive number.

password is a password of one to four alphanumeric characters with no blanks.

commands are Self-Contained Facility commands. Each command must end with a command terminator.

You can also use symbols for the following operators:

Operator	Symbol
EQ	=
GE	>= or => or ¬< or !<
GT	>
LE	<= or =< or ¬> or !>
LT	<
NE	¬= or !=
AND	&
OR	|
NOT	¬ or !

Without symbols, the software reads a blank on either side of an operator to separate it from other syntax units. However, with symbols, blanks are not required for separation. Any blanks found following a symbol are considered part of the value.

When the IF expression is true, the commands following THEN are processed. When the IF expression is false, the commands following ELSE are processed. Appropriate messages are issued to show the path taken. The IF expression accepts any combination of the AND, OR, and NOT operators similar to that of the QUEST language where-clause.

IF...THEN...ELSE statements can be nested up to 32 levels. The keyword ENDIF means that the current IF...THEN...ELSE sequence is complete. ENDIF is always required for each IF...THEN sequence.

The completion code reflects the results of processing a previous CONTROL, DEFINE, QUEST, REPORT, or QUEUE command. Valid completion code values are

- COMPLETE
  The previous command completed successfully. An informative message may also be issued.

- WARNING
  The previous command caused a warning message to be issued.

- ERROR
  The previous command caused a nonfatal, nondestructive message to be issued. In an interactive session, the previous command caused a batch-fatal, nondestructive message to be issued.

Messages that are currently fatal remain unchanged because existing jobs may depend on such fatalities. Messages that are batch-fatal but not fatal in an interactive session are treated as ERRORs interactively. The COMPLETE, WARNING, and ERROR completion codes coincide with the descriptions in *SYSTEM 2000 Messages and Codes, Version 12, First Edition*. Fatal (and batch-fatal) messages cause the job to terminate.

The completion code is set when a message is issued and is reset at the beginning of each command, with the following exceptions:

- an IF, ELSE, or ENDIF command

- a SHOW command

- a user comment

- a null statement (command terminator only).

If several messages result from a command, the completion code and message number of the message with the most severe completion code are maintained for testing. This completion code and its message number are written to the Message File if you use the SHOW command. See Chapter 47, "Recording IF...THEN...ELSE Variables: SHOW."

When testing for selected records, the value tested is the count of selected records resulting from the execution of a QUEST where-clause (sometimes displayed via message number -342-) or trace notation. The value can also be for a MAP where-clause in the DEFINE language or a GENERATE where-clause in the REPORT language.

If you specify MASTER PASSWORD (or MPW), the value of the condition is true if the password is the master password; if it is not the master password, the condition is false.

No syntax or logic verification occurs for commands skipped as a result of the software's evaluating the IF expression. The mechanism is similar to the skipping of command processing that follows a syntax error. Skipping can produce the following results:

- Commands skipped in one session may show errors in a subsequent session if a different logic path is followed.
- Skipping of erroneous syntax may cause errors in the software's determining which commands should be skipped.
- Skipping a sequence of commands always leaves you in the same processor. That is, if a command to change processors occurs within a sequence of commands being skipped, processors are not changed.
- User comments (if any) should follow a command terminator. User comments cannot be used within the IF *condition* THEN phrase; however, comments can occur after THEN or ELSE.
- No string or function expansion occurs during the skipping process.
- The Data File is not repositioned during skipping.

Considerations

- The EQ operator and = symbol are not interchangeable in an update action-clause. You must use EQ when specifying values in ADD, CHANGE, ASSIGN, and INSERT commands. However, if the value is a computation, you must use the = symbol.
- If component names contain any of the symbols, use the component numbers. Also, if you change the system separator to be any of the symbols, syntax errors could occur if you were to specify that symbol in IF...THEN...ELSE logic.
- If you are using Multi-User software, note the following:

    1. Do not imbed $LOCK or $END directives in IF-ENDIF blocks (that is, matching IF-ENDIF pairs). Use of these directives outside IF-ENDIF blocks is not affected.

    2. IF-ENDIF blocks must be completely contained in a single interactive session.

Examples

The following statements are examples of IF expression syntax, followed by examples of IF...THEN...ELSE logic.

```
IF COMPLETE THEN ...
IF WARNING OCCURRED AND MSG EQ 384 THEN ...
IF E OR W THEN ...
IF MESSAGE NUMBER = 507 THEN ...
IF MSG EQ 564 THEN ...
IF SELECTED RECORDS SPANS 8 * 25 THEN ...
IF RECS LT 4 THEN ...
IF MASTER PASSWORD THEN ...
IF PASSWORD EQ DEMO AND MSG = 547 THEN ...
IF MPW THEN ...
IF PW SPANS QA * QZ THEN ...
IF DB CYCLE = 12 THEN ...
```

Example 1

Suppose you do not know whether a database is damaged and must be restored for your job. The following message is issued if the database is damaged:

```
-810- WARNING: DATA BASE DAMAGED.  NO UPDATES PERMITTED.-
```

You can use the IF...THEN...ELSE logic shown here to test for message -810-. If message -810- appears, you can release the damaged database and restore it from tape. Notice that there are no commands for the ELSE portion in this example. If the database is not damaged, the logic goes directly to the ENDIF command. This command sequence ensures that the database is not in a damaged state.

```
USER, PASS:
DATA BASE NAME IS MYDATA:
IF MSG EQ 810 THEN
   CONTROL:
   RELEASE:
   RESTORE MYDATA FROM TAPE02:
ENDIF:
PRINT COUNT C1 WHERE ....
   .
   .
   .
```

Example 2

This example shows nesting to one level within the THEN portion of the sequence.

```
IF expression THEN IF expression THEN commands ELSE...ENDIF:
                  ELSE commands
                  ENDIF:
```

An alternate choice of indention for readability is

```
IF expression
THEN
   IF expression
   THEN
     commands
   ELSE
     commands
   ENDIF:
ELSE
  commands
ENDIF:
```

Chapter 47

Recording IF...THEN...ELSE Variables: SHOW

Format 47-1
Example 47-2

The SHOW command writes the following IF...THEN...ELSE system variables to the Message File:

- completion code
- message number
- database cycle number
- number of selected records.

The password is not written to the Message File. You can use the SHOW command within or outside an IF...THEN...ELSE sequence. For details about IF...THEN...ELSE, see Chapter 46, "Controlling Execution of Commands: IF...THEN...ELSE."

Format

```
SHOW SYSTEM VARIABLES  :
```

The completion code and message number are reset at the beginning of each Self-Contained Facility command, with the following exceptions:

- an IF, ELSE, or ENDIF command
- a SHOW command
- a user comment
- a null statement (the command terminator only).

Abbreviations:
SYSTEM = SYS
VARIABLES = VARS

The general format of SHOW command output is

```
-832- SYSTEM VARIABLE VALUES ARE AS FOLLOWS -
          COMPLETION CODE  = cc
          DATA BASE CYCLE  = cycle
          MESSAGE NUMBER   = msg
          SELECTED RECORDS = recs
```

cc = C, W, or E.

cycle = the database cycle.

msg = a message number.

recs = the last count of selected records.

If several messages result from a command, the completion code and message number of the message with the most severe completion code are maintained for testing. This completion code and its message number are written to the Message File.

Example

```
SHOW SYS VARS:

-832- SYSTEM VARIABLE VALUES ARE AS FOLLOWS -
      COMPLETION CODE  = C
      DATA BASE CYCLE  = 2942
      MESSAGE NUMBER   = 342
      SELECTED RECORDS = 37
```

Chapter 48

Exiting a SYSTEM 2000 Session: EXIT

Format 48-1
Example 48-1

The EXIT command ends a SYSTEM 2000 session and

- writes all updated database table pages to disk
- writes all Update Log pages, if any, to the Update Log disk file
- closes the database.

You issue the EXIT command as the last command in a session. You must issue an EXIT command in interactive mode to end a session. If an EXIT command is not included in a batch job or in an alternate Command File, the software automatically issues it.

If you need to exit a session right after invoking the SYSTEM 2000 software, you can issue the EXIT command as the first command in a session, which allows you to exit the session before issuing the USER command. That is, the first command in a session must be either the USER command or the EXIT command.

Format

```
EXIT :
```

Example

```
USER,ABLE:              (accesses SYSTEM 2000 software)
DBN IS EMPLOYEE:        (attaches the EMPLOYEE database)
   .
   . (QUEST commands)
   .
EXIT:                   (closes the database and the ends session)
```

Appendices

Guidelines for Avoiding Errors

Collating Sequence and Character Set

Special Characters

Reserved and Restricted Words and Characters

Reserved File Names

Appendix A

Guidelines for Avoiding Errors

PREVENTING COMMAND EXECUTION ERRORS A-1

PREVENTING UNDESIRABLE RESULTS A-2
Errors Caused by Previous Commands A-3
Errors Caused by Incorrect Command Usage A-5

This appendix presents a set of checklists for avoiding errors when you are retrieving or updating data.

PREVENTING COMMAND EXECUTION ERRORS

Basically, two types of errors prevent the execution of a command: syntax errors and semantic errors.

Syntax errors occur when the processor cannot recognize the command (for example, an UNLOAD command issued from the CONTROL processor) or when the command contains unacceptable or ambiguous syntax (for example, a missing comma before ORDERED BY).

When a syntax error occurs, the software displays a cursor along with a message. After encountering a syntax error, the software scans the Command File until it finds a command terminator; it does not detect subsequent syntax errors that could occur before the command terminator.

As part of its error processing, the software also inhibits string expansion. Therefore, it does not detect any command terminators coded within a string, but rather it continues to scan the Command File until it finds a command terminator. The software then proceeds with normal command processing, starting with the next syntax unit.

Semantic errors occur when the processor recognizes the command but cannot execute it because the command specifications conflict with processing rules. For example, the following command is syntactically correct:

```
PR C201 WH C102 EQ MARKETING:
```

However, because the target record (C200) and the focal record (C100) are disjoint, the software cannot execute the command. When a semantic error occurs, the software issues one or more messages indicating the nature of the problem. The cursor is not displayed.

Because commands are not executed when these types of errors occur, no damage is done to the database. However, the errors can be time-consuming to correct, especially in a batch environment. As explained in Chapter 26, "Optimization of Where-Clauses," there are cases when a syntax or semantic error in one command can cause subsequent commands to yield undesirable results.

The only way to prevent syntax or semantic errors is to ensure that commands are both syntactically and semantically correct before you issue them. The list below includes some of the things you should examine.

- Check for spelling in keywords, component names, and values.
- Check for missing or superfluous commas, asterisks, and so forth.
- Make sure that each command (except for $LOCK and $END) ends with a command terminator.
- In the case of retrieval or update commands, see whether the components specified need to be related. If so, include the HAS operator or a by-phrase.

PREVENTING UNDESIRABLE RESULTS

The problems discussed in this section apply more to update commands than to retrieval commands. If the output of a retrieval command is not what you had in mind, you can always correct and reissue the command. However, undesirable results from update commands are obviously more difficult to correct. In some cases, you can issue one or more update commands that undo the original command. Most often, though, you have to recover the database; that is, restore an archival copy of the database and apply the updates recorded in the Keepfile up to, but not including, the erroneous updates.

The best way to determine whether an update command has modified the database as desired is to issue a command that retrieves the items, records, or data trees affected. In the case of QUEST updates, you can use the SAME operator for this purpose. For example, you could issue the following update and retrieval commands:

```
ASSIGN C15 EQ AUSTIN TX* WH C1 EQ 1043:

PRINT C15 WH SAME:
```

After each update command, the software issues message -342-, which identifies the number of data records selected by the command. This number does not necessarily match the number of data records updated, but it indicates in general whether an update command modified the database as desired. If not, you should issue appropriate retrieval commands to determine which records were updated.

In the case of retrieval commands with a where-clause, the following message indicates that no data records were qualified by the where-clause, so none were selected.

```
-342-   0 SELECTED RECORDS (S) -
```

If no data are retrieved, the NULL format option is in effect, and message -342- is not displayed. You can then conclude that the database does not contain the data requested. For example, this command does not produce any output, because no C420 records exist for that entry.

```
PRINT C421 WHERE C1 EQ 1043:
```

There are basically two reasons why a command can give unexpected or undesirable results:

- One or more previous commands prevent the command from acting as expected.
- The command, although syntactically and semantically correct, does not express the desired request; that is, it is not used correctly.

Errors Caused by Previous Commands

Problems can occur when a previous command has set a certain option or limit or when previous commands contain syntax or semantic errors. The following list identifies some of these problems.

- Format options remain in effect even after you change processors. For example, if NULL SUPPRESS (the default) is in effect, the following command produces no output:

  ```
  PRINT C412 WHERE C1 EQ 1213:
  ```

  However, if issued with NULL in effect, the command displays a null item.

  ```
  PRINT /NULL/ C412 WHERE C1 EQ 1213:

  412* -NULL-
  ```

- The limits specified in a LIMIT command remain in effect until changed. These limits can affect the number of data records selected by a command, thus yielding unwanted results. On the other hand, you can use the LIMIT command to prevent some types of undesirable results from update commands. See Chapter 23, "Update Examples by Task" for examples.
- When an update (or retrieval) command depends on the correct execution of one or more prior update commands for its desired performance, you should ensure that the prior commands execute correctly before issuing the dependent command. This can often be done by using the LIMIT command or by using STOP IF or DISABLE EXECUTION commands.
- The absence of the command terminator can cause one or more subsequent commands to be in error or to be ignored. If only the three hyphens appear after a command, the problem could be that a command terminator was omitted in a previous command. The following sets of examples show possible problems.

```
PRINT C3 WH C2 EQ SMITH

PRINT C3 WH C2 EQ JONES:

-342-       0 SELECTED RECORD(S) -
```

Two PRINT commands are issued. However, because the first command does not end with a command terminator, the software processes the two commands as though they were one:

```
PRINT C3 WH C2 EQ SMITH PRINT C3 WH C2 EQ JONES:
```

Since there is no employee with LAST NAME (C2) equal to SMITH PRINT C3 WH C2 EQ JONES, no records qualify, so none are selected.

Now consider these commands:

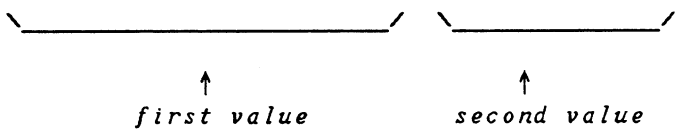
```
AS C126 EQ 3000.40* WH C2 EQ BOWMAN

AS C126 EQ 1557.00* WH C2 EQ CANADY:

- 342-       223 SELECTED RECORD(S) -
```

This example shows a more serious problem. Because the first command does not end with a command terminator, the software considers the second command to be a continuation of the first. Thus, it takes the EQ operator to be a ternary operator and does the update for each employee whose last name falls between BOWMAN AS C126 EQ 1557.00 and WH C2 EQ CANADY, that is, almost all employees. Notice that the asterisk in the second command makes the software treat EQ as equivalent to the SPANS operator. The command executed by the software is

```
AS C126 EQ 3000.40* WH C2 EQ BOWMAN AS C126 EQ 1557.00* WH C2 EQ CANADY:
```



Errors Caused by Incorrect Command Usage

The following list identifies some of the usage errors that cause an error-free command to yield unwanted results:

- Misuse of operators in where-clauses:

  a) The HAS operator can cause actions to take place in more data records than you intend. For example, the following command retrieves C111 values from all those C110 records that are descendants of C0 records, which in turn are ancestors of C110 records with C111 values greater than or equal to 1800.

  ```
  PR C111 WH C0 HAS C111 GE 1800.00:
  ```

  For instance, it retrieves values for C111 from records 38, 14, 15, 17 in Logical Entry Two (shown in the foldout at the end of this manual). Notice that the values C111 in records 15 and 17 are less than 1800. The command should be

  ```
  PR C111 WH C111 GE 1800.00:
  ```

  b) Parentheses used in combination with the logical operators AND and OR can cause problems if they are not used correctly. For example, the command

  ```
  PR C2 WH C412 = MBA & MAJOR FIELD = MARKETING OR
     MINOR FIELD = MARKETING:
  ```

  gives a different result than the following command:

  ```
  PR C2 WH C412 = MBA & (MAJOR FIELD EQ MARKETING OR
     MINOR FIELD = MARKETING):
  ```

- In the case of retrieval commands, using by-phrases can cause a problem similar to the one mentioned above in reference to the HAS operator, that is, retrieval of data from more records than desired.

- SAME, DITTO, and PREVIOUS can cause problems if not used properly. For example, consider the following commands:

  ```
  ASSIGN C414 EQ COMPUTER SCIENCE* WH C1 EQ 1031:

  ASSIGN C415 EQ PREVIOUS* WH C1 EQ 1266:
  ```

  Because an asterisk is specified after PREVIOUS, the software treats the keyword PREVIOUS as a value and not as an operator. Therefore, it assigns the value PREVIOUS to C415. The second command should be

  ```
  ASSIGN C145 EQ PREVIOUS WH C1 EQ 1266:
  ```

Appendix B

Collating Sequence and Character Set

The ascending collating sequence for the EBCDIC (Extended Binary Coded Decimal Interchange Code) character set is provided in this appendix.

Collating Sequence	Hexadecimal Representation	Symbol	Meaning
. . .			
64	40		Space
. . .			
74	4A	¢	Cent sign
75	4B	.	Period, decimal point
76	4C	<	Less than sign
77	4D	(	Left parenthesis
78	4E	+	Plus sign
79	4F	|	Vertical bar, logical OR
80	50	&	Ampersand
. . .			
90	5A	!	Exclamation point
91	5B	$	Dollar sign
92	5C	*	Asterisk
93	5D	)	Right parenthesis
94	5E	;	Semi-colon
95	5F	¬	Logical NOT
96	60	-	Minus sign, hyphen
97	61	/	Slash
. . .			
107	6B	,	Comma
108	6C	%	Percent sign
109	6D	_	Underscore
110	6E	>	Greater than sign
111	6F	?	Question mark
. . .			
122	7A	:	Colon
123	7B	#	Number sign
124	7C	@	At sign

(continued)

Collating Sequence	Hexadecimal Representation	Symbol	Meaning
125	7D	'	Apostrophe
126	7E	=	Equals sign
127	7F	"	Quotation marks
.			
.			
.			
193	C1	A	
194	C2	B	
195	C3	C	
196	C4	D	
197	C5	E	
198	C6	F	
199	C7	G	
200	C8	H	
201	C9	I	
.			
.			
.			
209	D1	J	
210	D2	K	
211	D3	L	
212	D4	M	
213	D5	N	
214	D6	O	
215	D7	P	
216	D8	Q	
217	D9	R	
.			
.			
.			
226	E2	S	
227	E3	T	
228	E4	U	
229	E5	V	
230	E6	W	
231	E7	X	
232	E8	Y	
233	E9	Z	
.			
.			
.			
240	F0	0	
241	F1	1	
242	F2	2	
243	F3	3	
244	F4	4	
245	F5	5	
246	F6	6	
247	F7	7	
248	F8	8	
249	F9	9	

Appendix C

Special Characters

You can use these special characters in SYSTEM 2000 commands:

¢	$	_
.	*	>
<	)	?
(	;	#
+	¬	@
|	-	'
&	/	=
!	%	"

Appendix D

Reserved and Restricted Words and Characters

You cannot use these reserved words and characters in component names, even when they are preceded and followed by special characters. (For example, -OR- is not acceptable.)

Words That Cannot Begin a Component Name

ALL
ANY
AVG
C (as a single letter)
C*nnnn*
COUNT
DATA
HIGH
LOW
MAX
MIN
NK
NON-KEY
OB
ORDERED BY
SA
SAME
SIGMA
SUM
TREE

Words and Characters That Cannot Occur in a Component Name

AND
AT
BY
CONT
CONTAIN
CONTAINING
CONTAINS
CTNSIN
ELSE
EQ
EXIST
EXISTING
EXISTS
FAIL
FAILING
FAILS
FROM
GE
GT
HAS
HAVE
HAVING
IN
LE
LT
MOVE
NE
NOT
OCCURS
OR
SPAN
SPANNING
SPANS
TO
WH
WHERE
,
(
)
system separator
command terminator

In addition, you cannot use the following operator symbols in component names:

=
>=
=>
¬<
!<
<=
=<
¬>
!=
&
|
¬
!

You can use the following words in component names, but you must reference the component in a command by its component number.

Restricted Usage To Begin a Component Name

CNT
OF
RCNT
RESUM

Restricted Usage Anywhere in a Component Name

FOR
THROUGH
THRU

Appendix E

Reserved File Names

Following are the reserved file names:

S2KSNAP	S2KSYS00	SF01
S2KMSG	S2KSYS01	SF02
S2KCOMP	S2KSYS02	SF03
SYSUDUMP	S2KSYS03	SF04
SYSABEND	S2KSYS04	SF05
SYSOUT	S2KSYS05	SF06
S2KUSERS	S2KSYS06	
TAPES2K	S2KSYS07	SF11
KEEPFILE	S2KSYS08	SF12
STEPLIB		SF13
SYSLIB	S2KSYS10	.
S2KPARMS	S2KSYS11	.
S2KPAD*nn*	S2KSYS12	.
	.	and so forth
	.	
	.	
	and so forth	

In addition, the following DDnames are reserved:

- the eight DDnames used for the database files to be accessed in the same SYSTEM 2000 session
- the two DDnames used for the Savefile and Keepfile.

A DDname consists of the first seven characters of the database name, plus a suffix. The suffix is a number from 1 through 8 for the database files, a letter S or K for the Savefile or Keepfile. If the database name is less than seven characters long, the DDname uses Xs to fill it out to seven characters. For example, if the database name is AUTO, the DDname for the Savefile is AUTOXXXS.

Index

Special Characters
$LOCK option 32-9
* (asterisk), default system separator 87
FTODAY system string 75, 30-2
NOW system string 75, 30-1
TODAY system string 75, 30-1
/ (slash), default where-clause delimiter 87

A
action-clauses 24, 39
    repeating with DITTO 32-2
ad hoc functions 74, 33-1
ADD command 54, 20-6
adding new data trees, INSERT TREE command 61, 21-15
allocating user files dynamically 38-3
alternate user files
    Command File 38-5
    Data File 38-8
    Message File 38-13
    Report File 38-17
ancestor record 7
AND logical operator in a where-clause 26, 24-25
arithmetic statistics, obtaining with system functions 72, 34-1
ASSIGN command 59, 20-10
ASSIGN TREE command 66, 21-4
assigning values within existing data records, ASSIGN command 59, 20-10
asterisk (*), default system separator 87
AT logical operator in a where-clause 29, 24-33
AUTHORITIES option in DESCRIBE command 28-4
    with DEFINE option 28-5
authorities, displaying with DESCRIBE command 28-4
    with DEFINE output 28-5
AVG system function 72, 34-2

B
binary operators in a where-clause 25
    comparing an item with a value 24-9
    comparing item with Collect File item 24-12
    comparing two items 24-11
BLOCK format option 15-4
by-clause 50, 17-1
by-phrase in a by-clause 50, 17-2

C
C-number 5
calculations, obtaining with ad hoc functions 74, 33-1
calling CONTROL, DEFINE, and QUEST languages 37-1
calling QUEST language 82, 37-1

CHANGE command 56, 20-15
changing processors 83, 37-1
CHARACTER item type
    coding values for loading 8-16
    defining 11
character set B-1
characters
    reserved and restricted D-1
    special C-1
child record 7
CLEAR commands 70, 31-19
collating sequence B-1
COLLECT command 45, 10-1
    processing 10-5
Collect File 45
    creating with COLLECT command 10-1
    displaying items with DESCRIBE command 28-3
    example 10-7
    item names 10-4
    item picture 10-5
    unloading 14-3
    using 10-7
    where-clause processing 24-13
column heading specification in LIST command 11-10
column width specification in LIST command 11-19
columnar format retrieval, LIST command 11-1
Command File 88
    allocating dynamically 38-3
    description 38-1
    switching 38-5
COMMAND FILE IS command 88, 38-5
command processing, controlling 70, 31-1
command units, repeating 71, 32-1
commands, issuing 84
comments, inserting 9-9, 44-1
components 5
    displaying with DESCRIBE command 28-3
conditional execution of CONTROL, DEFINE, and QUEST commands 46-1
conditions in a where-clause 24, 24-6
    controlling with parentheses 36, 24-37
    processing order 36
CONTAINS operator, searching for characters in an item 25, 24-17
CONTINUE commands 70, 31-15
CONTROL command 37-1
control nodes 9
control records 9
controlling processing commands 70, 31-1
    CLEAR 31-19
    ENABLE/DISABLE EXECUTION 31-18
    LIMIT/END LIMIT 31-2
    STOP/CONTINUE 31-15
controlling the execution of CONTROL, DEFINE, and QUEST commands 46-1

COPY TREE command 68, 21-10
copying trees, COPY TREE command 68, 21-10
COUNT system function 72, 34-2
CTNSIN operator, implying OR between CONTAINS conditions 24-23
cycle number 12

D
data entry *See* logical entry
Data File 88
  allocating dynamically 38-3
  coding values for loading 8-12
  description 38-1
  preparing for loading 8-10
  switching 38-8
DATA FILE IS command 88, 38-8
data records 3
  updating 20-1
Data Table 15, 17
data trees 3
  updating 61, 21-1
data values, updating 54
database 2
  cycle number 12
  defining 10
  displaying definition 12
  Index 15
  loading 14
  name 12
  physical view 15
  reloading 80
  schema 4
  tables 15
DATE FORMAT IS command 27-1
date format, changing 27-1
DATE item type
  coding values for loading 8-15
  defining 11
date system strings 75, 30-1
DECIMAL item type
  coding values for loading 8-13
  defining 11
DEFINE command 37-1
DEFINE option in DESCRIBE command 28-4
  with AUTHORITIES option 28-5
DEFINE output, producing with DESCRIBE command 28-4
  with password authorities 28-5
defining
  database 10
  schema items 10
  schema records 10
  stored strings and functions 11
definition number 12

Definition Table 15
DELIMITER IS command 87, 41-1
delimiter, where-clause 87
descendant record 7
DESCRIBE command 12, 28-1
DISABLE EXECUTION command 70, 31-18
DISABLE VALUES PADDING command 16, 29-1
disjoint records 9
displaying database definition 12
    DESCRIBE command 28-1
displaying values together using by-clause 50, 17-1
dissimilar records 5
Distinct Values Table 15
    padding with ENABLE/DISABLE VALUES PADDING command 29-1
DITTO operator 71, 32-2
DOUBLE item type
    coding values for loading 8-14
    defining 11
DOUBLE SPACE format option 15-5
dynamic allocation of user files 38-3

E
ECHO command 43-1
ENABLE EXECUTION command 70, 31-18
ENABLE VALUES PADDING command 16, 29-1
END FRAME command 22-1
END LIMIT command 31-2
END, default entry terminator word 87
ending a session 82
    EXIT command 48-1
entering software and QUEST language 82
ENTRY record 3, 7
entry terminator 87
    changing with ENTRY TERMINATOR IS command 39-1
ENTRY TERMINATOR IS command 87, 39-1
environments, processing 86
EQ operator, comparing an item with a range of values 24-15
Errors 88
    during loading 8-19
    guidelines for avoiding A-1
examples by task
    retrieval commands 18-1
    update commands 23-1
execution
    controlling in event of errors with STOP/CONTINUE commands 31-15
    preventing with ENABLE/DISABLE EXECUTION commands 31-18
EXISTS operator 24-7
EXIT command 48-1
exiting software 82, 48-1
expressions, in where-clause 26, 24-23, 24-25
Extended Field Table 15

F
FAILS operator 24-8
family of a record 8
files, reserved names E-1
focal record 34
  specifying for system function 34-6
  specifying in where-clause with HAS operator 24-29
format options for retrieval commands 48, 15-1
  FORMOP command, changing with 15-22
  HEX OFF/HEX ON 15-21
  INDENT/BLOCK 15-4
  NULL SUPPRESS/NULL 15-14
  NUMBER/NAME 15-10
  REPEAT/REPEAT SUPPRESS 15-18
  SINGLE SPACE/DOUBLE SPACE 15-5
  STUB/STUB SUPPRESS 15-12
  TREE/RECORD 15-17
  ZERO SUPPRESS/ZERO 15-15
FORMOP command 15-22
FRAME command 22-1
full trace 20-4
functions
  ad hoc 74, 33-1
  stored
    defining 11
    displaying with DESCRIBE command 28-3
    invoking 75
    parametric, invoking 35-12
    simple, invoking 35-10
  system 72, 34-1

G
GE operator
  comparing an item with a Collect File item 24-12
  comparing an item with a value 24-9
  comparing two items 24-11
GT operator
  comparing an item with a Collect File item 24-12
  comparing an item with a value 24-9
  comparing two items 24-11
guidelines
  for avoiding errors A-1
  for issuing QUEST commands 84

H
HAS logical operator in a where-clause 32, 24-29
HEX OFF format option 15-21
HEX ON format option 15-21
hierarchical structure 3
Hierarchical Table 15, 17

I

IF...THEN...ELSE commands 46-1
IN operator, implying OR between EQ conditions 24-23
INDENT format option 15-4
Index 15
  padding with ENABLE/DISABLE VALUES PADDING command 16, 29-1
INSERT TREE command 61, 21-15
inserting comments 9-9, 44-1
INTEGER item type
  coding values for loading 8-13
  defining 11
invoking
  stored functions
    parametric 35-12
    simple 35-10
  stored strings
    parametric 35-6
    simple 35-2
issuing QUEST commands, guidelines for 84
item picture 10
item types 10
items
  Collect File 10-4
  null 9

J

JCL, alternate user files 38-4

K

key conditions in a where-clause 37, 24-24
key item 10
key value retrieval, TALLY command 47, 13-1
key-processing 37

L

layout specifications in LIST command 11-4
LE operator
  comparing an item with a Collect File item 24-12
  comparing an item with a value 24-9
  comparing two items 24-11
levels in a database 8
LIMIT command 70, 31-2
LIST command 43, 11-1
  column heading specification 11-10
  column width specification 11-19
  general format 11-2
  layout specifications 11-4
  multiple-line heading specification 11-16
  page width specification 11-6
  S-specification 11-25
    repeated headings 11-31
    spacing between rows 11-28
    suppressed row identifiers 11-31
  title and paging specification 11-6

LOAD command 8-1
loader stream format 14, 8-10
loading a database 14, 7-1
  coding values for Data File 8-12
  efficiency considerations 8-18
  interactively 8-8
  LOAD command 8-1
  methods of 7-1
  preparing the Data File 8-10
  problems, typical 8-19
  processing 8-24
  selecting a method of 7-3
  steps 8-5
  techniques 8-6
logical entry 3
logical operators 26, 24-25
  AND 26, 24-25
  AT 29, 24-33
  controlling with parentheses 36, 24-37
  HAS 32, 24-29
  NOT 27, 24-25
  OR 27, 24-25
  processing order 36
  processing order of conditions 24-37
logical unit of work, creating with FRAME and END FRAME commands 22-1
LOOKUP IS command 37-3
LT operator
  comparing an item with a Collect File item 24-12
  comparing an item with a value 24-9
  comparing two items 24-11

M

Master Record 15
MAX system function 72, 34-2
Message File 88
  allocating dynamically 38-3
  description 38-1
  switching 38-13
MESSAGE FILE IS command 88, 38-13
Messages 88
MIN system function 72, 34-2
MONEY item type
  coding values for loading 8-14
  defining 11
monitoring performance 45-1
MOVE TREE command 67, 21-37
moving trees, MOVE TREE command 67
Multi-User environment considerations 86
Multiple Occurrence Table 15
multiple-line heading specification in LIST command 11-16

N

NAME format option 15-10
NE operator, comparing an item with a range of values 24-15
non-key conditions in a where-clause 37, 24-24
non-key processing 37
normalization in a where-clause 33, 25-1
NOT logical operator in a where-clause 27, 24-25
NULL format option 15-14
null items and records 9
NULL SUPPRESS format option 15-14
nulls
    replacing values with 20-19
    replacing with values 20-6
NUMBER format option 15-10

O

opening database 82
operators in a where-clause *See* Where-clause
optimization considerations
    retrieval commands 9-8
    update commands 19-2
    where-clauses 26-1
OR logical operator in a where-clause 27, 24-25
ordering output *See* ordering-clause
ordering-clause 49, 16-1
    sort keys 49
    sort-keys 16-3
output
    formats, changing with format options 48, 15-1
    order, changing with ordering-clause 49, 16-1
    volume, controlling with LIMIT/END LIMIT commands 70, 31-2

P

padding 16
    with ENABLE/DISABLE VALUES PADDING command 29-1
page width specification in LIST command 11-6
paging specification in LIST command 11-6
parametric functions 75, 35-12
parametric strings 75, 35-6
parent record 7
parentheses in where-clause 36, 24-37
partial trace 20-5
password authorities, displaying with DESCRIBE command 28-4
    and DEFINE output 28-5
password, specifying 82
path of a record 8
physical view of a database 15
picture of an item 10
position, specifying in where-clause with AT logical operator 29, 24-33
predicate operators, IN and CTNSIN 24-23
PREVIOUS operator 72, 32-4
PRINT command 41, 12-1

processing environments 86
processing order of where-clause conditions 36, 24-37
processor lookup 37-3
processor prompt feature 42-1
processors
    changing 83, 37-1
    changing automatically 37-3
PROMPT IS command 42-1
prompt, changing 42-1

Q
qualification criteria, repeating with SAME 32-5
qualified data records 26
qualifying data see where-clause
QUEST command 37-1
QUEST language, calling 82, 37-1
QUEUE command 37-1

R
REAL item type
    coding values for loading 8-14
    defining 11
record definition, displaying with DESCRIBE command 28-5
RECORD format option 15-17
RECORD option in DESCRIBE command 28-5
records
    control 9
    disjoint 9
    dissimilar 5
    null 9
    related 9
    relationships among 7
    similar 5
related records 9
relational table *See* Collect File
relationships among records in a tree 7
release number 12
RELOAD command 80
REMOVE command 58, 20-19
REMOVE TREE command 64, 21-42
removing records and descendants, REMOVE TREE command 64, 21-42
REPEAT format option 15-18
REPEAT SUPPRESS format option 15-18
repeated headings in LIST command S-specification 11-31
repeating
    action-clauses with DITTO 32-2
    command units 71, 32-1
    qualification criteria with SAME 32-5
    value streams with PREVIOUS operator 32-4
    where-clauses, specifying with SAME IS DYNAMIC/STATIC 32-7
repetitive processing 71, 32-1
    DITTO 32-2
    Multi-User environment considerations 32-9
    PREVIOUS 32-4

SAME 32-5
SAME IS DYNAMIC/STATIC 32-7
replacing
existing data trees, ASSIGN TREE command 66, 21-4
nulls with values, ADD command 54, 20-6
values with new values, CHANGE command 56, 20-15
values with nulls, REMOVE command 58, 20-19
REPORT command 37-1
Report File 88
allocating dynamically 38-3
description 38-1
switching 38-17
REPORT FILE IS command 88, 38-17
reserved
file names E-1
words and characters D-1
restricted words and characters D-1
retrieval commands 41, 9-1
by-clause 50, 17-1
changing output formats 15-1
COLLECT 10-1
comments, inserting 9-9
considerations 9-3
examples by task 18-1
format options 48
LIST 11-1
optimization considerations 9-8
ordering-clause 16-1
PRINT 12-1
processing 9-4
retrieval-clause 40
TALLY 13-1
UNLOAD 14-1
retrieval objects 40
retrieval-clause 9-2
reusable space 18
root of a data tree 3

S

S-specification in LIST command 11-25
repeated headings 11-31
spacing between rows 11-28
suppressed row identifiers 11-31
SAME IS DYNAMIC/STATIC command 32-7
SAME operator 71, 32-5
schema
components 5
database 4
items 4
defining 10
records 4
defining 10
trees 4

searching for characters in an item, CONTAINS operator 25, 24-17
security considerations 85
SEPARATOR IS command 87, 40-1
sequential retrieval, PRINT command 41, 12-1
session, ending 82, 48-1
SHOW command 47-1
sibling records 7
SIGMA system function 72, 34-3
similar records 5
simple functions 75, 35-10
simple strings 75, 35-2
SINGLE SPACE format option 15-5
single-user environment 86
slash (/), default where-clause delimiter 87
sort keys in ordering-clause 49, 16-3
space, reusable 18
spacing between rows in LIST command S-specification 11-28
SPANS operator, comparing an item with a range of values 24-15
special characters C-1
STOP commands 70, 31-15
stored functions *See* Functions
stored strings *See* Strings
strings 75
    defining 11
    displaying with DESCRIBE command 28-3
    parametric, invoking 35-6
    simple, invoking 35-2
    system 75, 30-1
STUB format option 15-12
STUB SUPPRESS format option 15-12
subtree 8
SUM system function 72, 34-2
suppressed row identifiers in LIST command S-specification 11-31
synchpoints, establishing with FRAME and END FRAME commands 22-1
system functions 72, 34-1
    AVG 34-2
    COUNT 34-2
    focal record, specifying 34-6
    MAX 34-2
    MIN 34-2
    SIGMA 34-3
    SUM 34-2
system prompt, changing 42-1
system release number 12
system separator 87
    changing with SEPARATOR IS command 40-1
system strings 75, 30-1
    FTODAY 30-2
    NOW 30-1
    TODAY 30-1
system-wide commands
    affecting QUEST language 87
    COMMAND FILE IS 38-5
    CONTROL 37-1

DATA FILE IS 38-8
DEFINE 37-1
DELIMITER IS 41-1
ECHO 43-1
ENTRY TERMINATOR IS 39-1
EXIT 48-1
IF...THEN...ELSE 46-1
inserting comments 44-1
LOOKUP IS 37-3
MESSAGE FILE IS 38-13
PROMPT IS 42-1
QUEST 37-1
REPORT FILE IS 38-17
SEPARATOR IS 40-1
SHOW 47-1
TIMING 45-1
S2KOUTP intermediate file 38-15, 38-19
dynamically allocating 38-15

T

tables, database 15
TALLY command 47, 13-1
target record, specifying
in a retrieval-clause 9-4
in an update command 53
ternary operators in a where-clause 25, 24-15
TEXT item type
coding values for loading 8-16
defining 11
text search operator CONTAINS 25, 24-17
time system string 75, 30-1
TIMING command 45-1
title specification in LIST command 11-6
trace notation, specifying in an update command 53, 20-3
TREE format option 15-17

U

unary operators in a where-clause 25
EXISTS 24-7
FAILS 24-8
UNDEFINED item type
coding values for loading 8-17
controlling output format 15-21
defining 11
UNLOAD command 46, 14-1
unloading a Collect File 14-3
update commands 53, 19-1
ADD 20-6
ASSIGN 20-10
ASSIGN TREE 21-4
CHANGE 20-15
COPY TREE 21-10
examples by task 23-1
FRAME and END FRAME 22-1

INSERT TREE 21-15
MOVE TREE 21-37
optimization considerations 19-2
REMOVE 20-19
REMOVE TREE 21-42
target record, specifying 53
trace notation 53, 20-3
updated pages, moving to disk with CLEAR commands 70, 31-19
updating data items 54
updating data records 20-1
ADD command 20-6
ASSIGN command 20-10
CHANGE command 20-15
REMOVE command 20-19
updating data trees 61, 21-1
ASSIGN TREE command 21-4
INSERT TREE command 21-15
MOVE TREE command 21-37
REMOVE TREE command 21-42
user files 88, 36-1, 38-2
allocating dynamically 38-3
alternate user files 38-1
JCL 38-4
LOCAL option 38-4

V

value streams
repeating with PREVIOUS operator 32-4
retrieving with UNLOAD command 14-1
values 3
updating 54
values padding *See* padding

W

where-clause 24, 24-1
AND logical operator 26, 24-25
AT logical operator 29, 24-33
binary operators 25
comparing an item with a Collect File item 24-12
comparing an item with a value 24-9
comparing two items 24-11
Collect File processing 24-13
conditions 24, 24-6
controlling with parentheses 36, 24-37
processing order 36, 24-37
considerations 24-5
CONTAINS operator 25, 24-17
CTNSIN operator 24-23
delimiter 87
changing 41-1
EXISTS operator 24-7
expressions 26, 24-25
FAILS operator 24-8
focal records, specifying with HAS operator 34, 24-29

HAS logical operator 32, 24-29
IN operator 24-23
key and non-key conditions in 37, 24-24
logical operators 26, 24-25
AND 26, 24-25
AT 29, 24-33
HAS 32, 24-29
NOT 27, 24-25
OR 27, 24-25
normalization 33, 25-1
NOT logical operator 27, 24-25
operators 25
optimization considerations 26-1
OR logical operator 27, 24-25
parentheses 36, 24-37
position, specifying with AT logical operator 29
processing order of conditions 36, 24-37
qualified data records 26
specifying which to repeat with SAME IS DYNAMIC/STATIC 32-7
ternary operators 25, 24-15
text search operator CONTAINS 25, 24-17
unary operators 25
EXISTS 24-7
FAILS 24-8
width of page, LIST command 11-6
words, reserved and restricted D-1

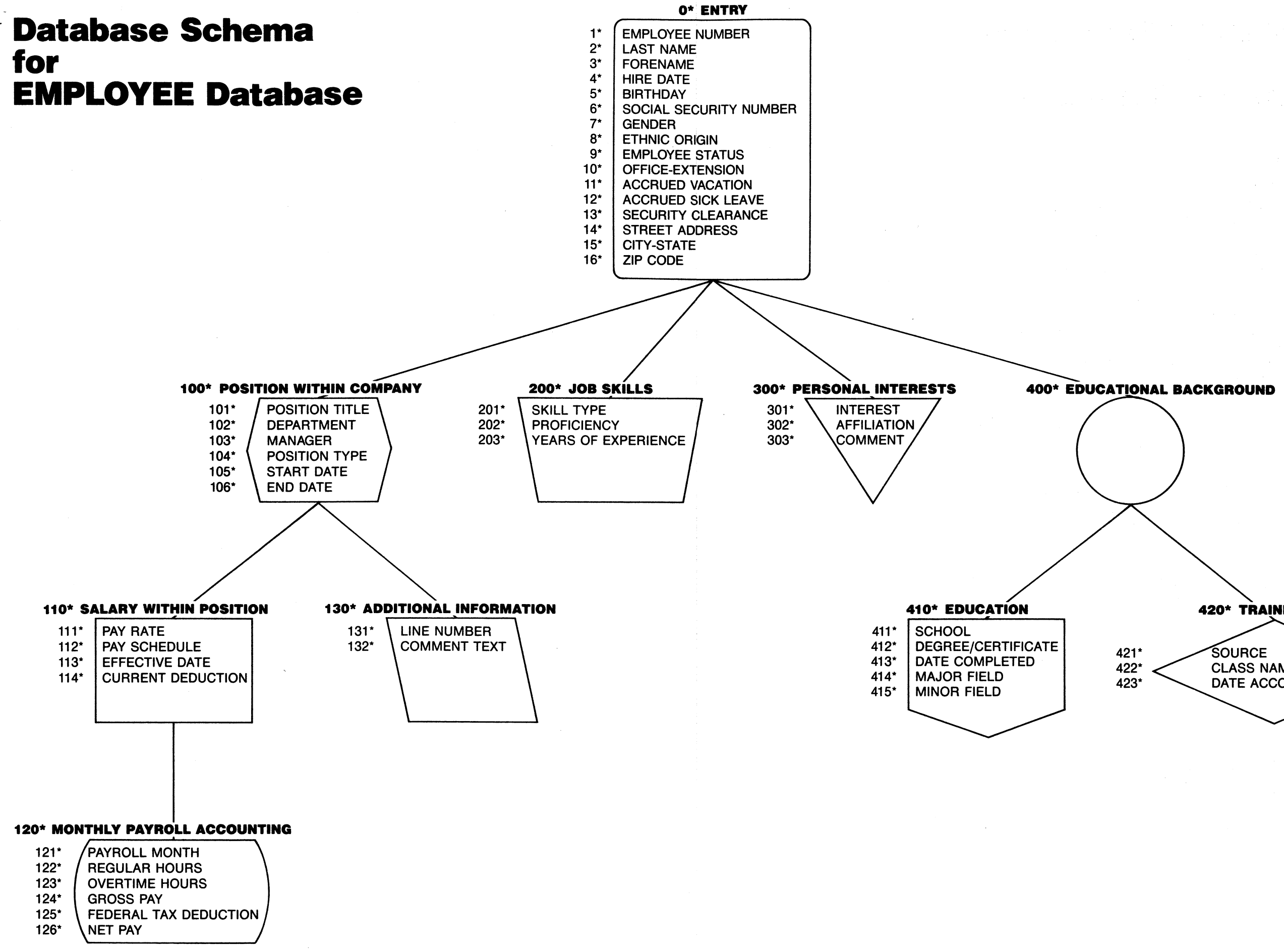
Z

ZERO format option 15-15
ZERO SUPPRESS format option 15-15

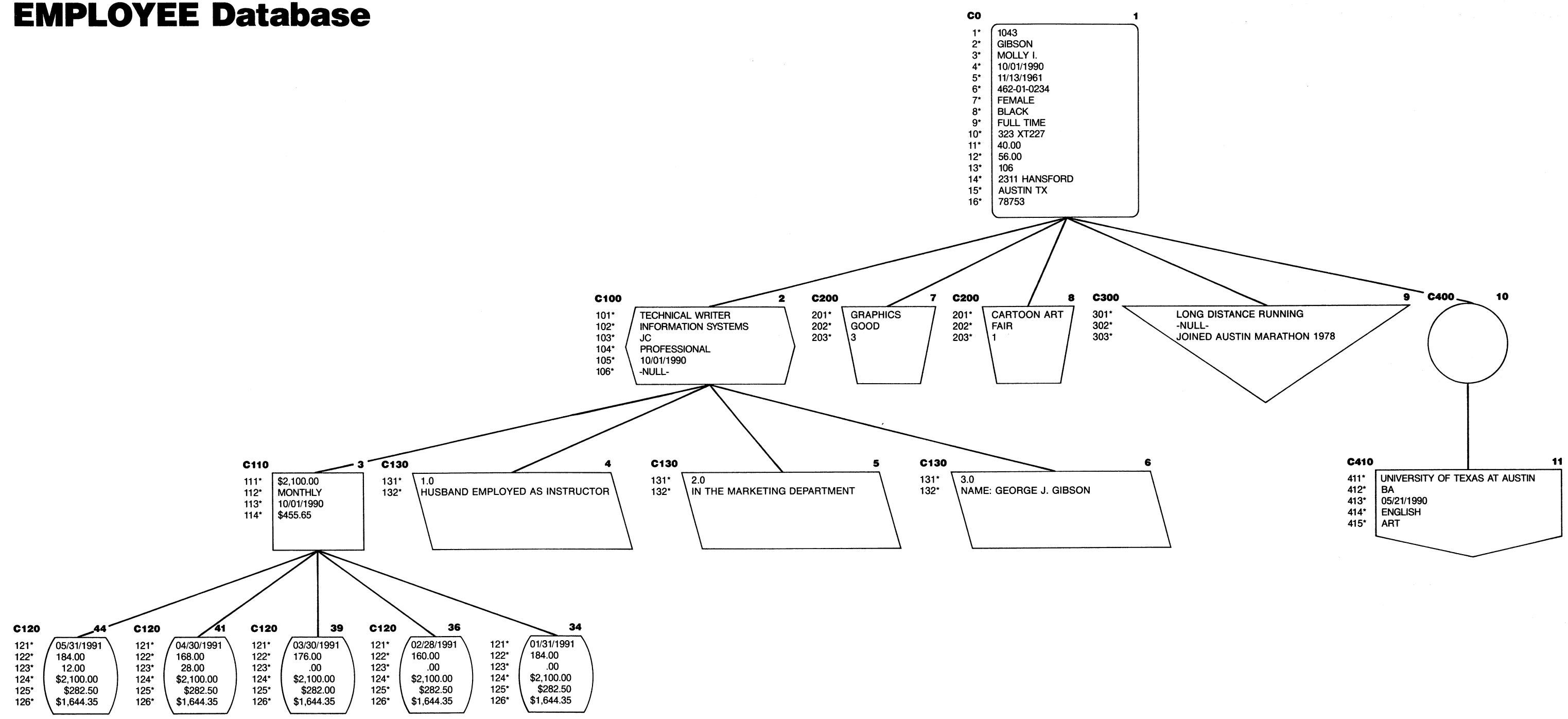
Actual DESCRIBE Output from EMPLOYEE Database

```
DESCRIBE:
SYSTEM RELEASE NUMBER XXX
DATABASE NAME IS          EMPLOYEE
DEFINITION NUMBER                 3
DATA BASE CYCLE NUMBER            1
      1*   EMPLOYEE NUMBER (INTEGER NUMBER 9999)
      2*   LAST NAME (CHAR X(10) WITH FEW FUTURE OCCURRENCES)
      3*   FORENAME (NON-KEY CHAR X(20))
      4*   HIRE DATE (DATE)
      5*   BIRTHDAY (DATE)
      6*   SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11))
      7*   GENDER (CHAR X(6) WITH MANY FUTURE OCCURRENCES)
      8*   ETHNIC ORIGIN (CHAR X(9) WITH MANY FUTURE OCCURRENCES)
      9*   EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES)
     10*   OFFICE-EXTENSION (NON-KEY CHAR X(9))
     11*   ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99)
     12*   ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99)
     13*   SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES)
     14*   STREET ADDRESS (NON-KEY CHAR X(20))
     15*   CITY-STATE (NON-KEY CHAR X(15))
     16*   ZIP CODE (CHAR X(5) WITH FEW FUTURE OCCURRENCES)
    100*   POSITION WITHIN COMPANY (RECORD)
      101*    POSITION TITLE (NON-KEY CHAR X(10) IN 100)
      102*    DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES)
      103*    MANAGER (CHAR XXX IN 100 WITH FEW FUTURE OCCURRENCES)
      104*    POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCES)
      105*    START DATE (DATE IN 100)
      106*    END DATE (NON-KEY DATE IN 100)
      110*    SALARY WITHIN POSITION (RECORD IN 100)
        111*     PAY RATE (MONEY $9999.99 IN 110)
        112*     PAY SCHEDULE (CHAR X(7) IN 110)
        113*     EFFECTIVE DATE (DATE IN 110)
        114*     CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)
        120*     MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)
          121*      PAYROLL MONTH (DATE IN 120)
          122*      REGULAR HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
          123*      OVERTIME HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
          124*      GROSS PAY (NON-KEY MONEY $9999.99 IN 120)
          125*      FEDERAL TAX DEDUCTION (NON-KEY MONEY $9999.99 IN 120)
          126*      NET PAY (NON-KEY MONEY $9999.99 IN 120)
      130*    ADDITIONAL INFORMATION (RECORD IN 100)
        131*     LINE NUMBER (DECIMAL NUMBER 99.9 IN 130)
        132*     COMMENT TEXT (NON-KEY TEXT X(7) IN 130)
  200*    JOB SKILLS (RECORD)
      201*    SKILL TYPE (CHAR X(12) IN 200 WITH SOME FUTURE OCCURRENCES)
      202*    PROFICIENCY (NON-KEY CHAR X(5) IN 200)
      203*    YEARS OF EXPERIENCE (NON-KEY INTEGER NUMBER 99 IN 200)
  300*    PERSONAL INTERESTS (RECORD)
      301*    INTEREST (CHAR X(12) IN 300 WITH FEW FUTRUE OCCURRENCES)
      302*    AFFILIATION (NON-KEY CHAR X(5) IN 300)
      303*    COMMENT (NON-KEY TEXT X(5) IN 300)
  400*    EDUCATIONAL BACKGROUND (RECORD)
      410*    EDUCATION (RECORD IN 400)
        411*     SCHOOL (CHAR X(15) IN 410)
        412*     DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH FEW FUTURE OCCURRENCES)
        413*     DATE COMPLETED (DATE IN 410)
        414*     MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
        415*     MINOR FIELD (NON-KEY CHAR X(12) IN 410)
      420*    TRAINING (RECORD IN 400)
        421*     SOURCE (NON-KEY CHAR X(12) IN 420)
        422*     CLASS NAME (CHAR X(12) IN 420 WITH FEW FUTURE OCCURRENCES)
        423*     DATE ACCOMPLISHED (DATE IN 420)
```

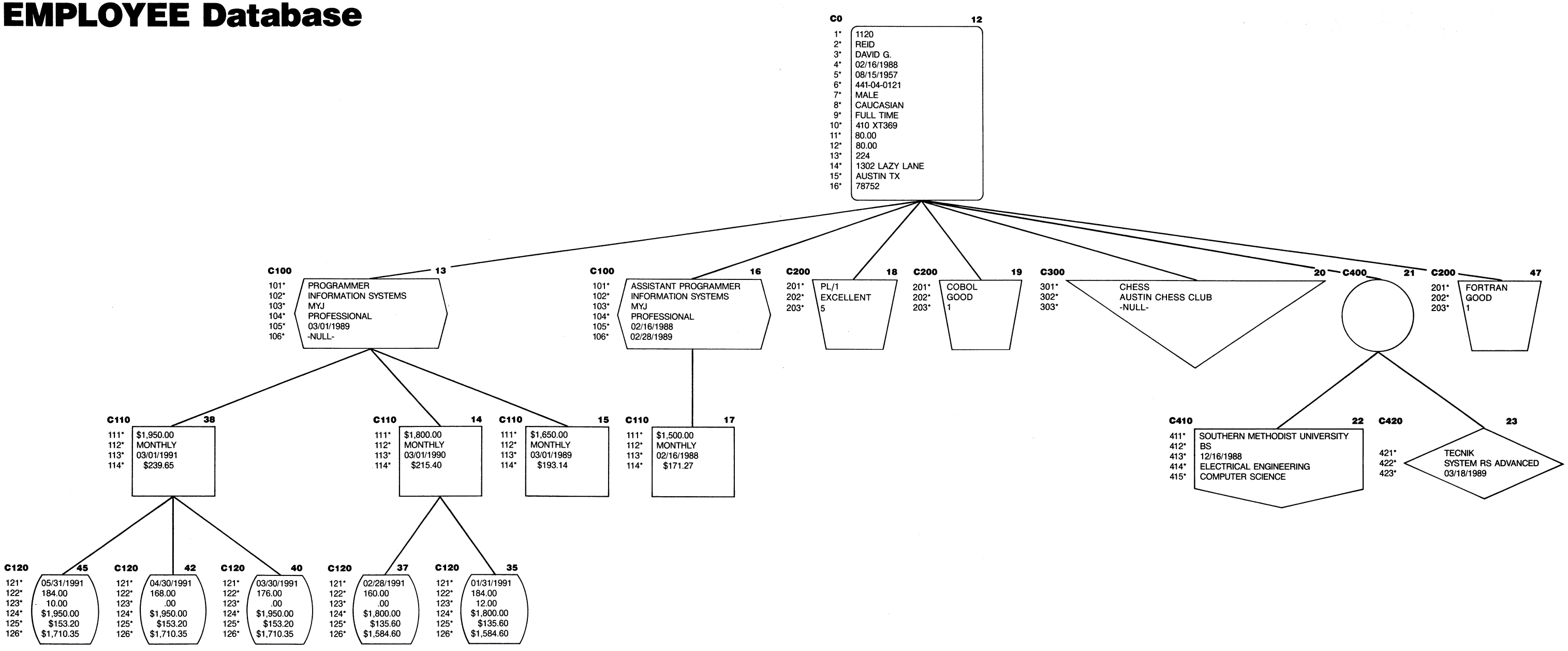
Database Schema for EMPLOYEE Database



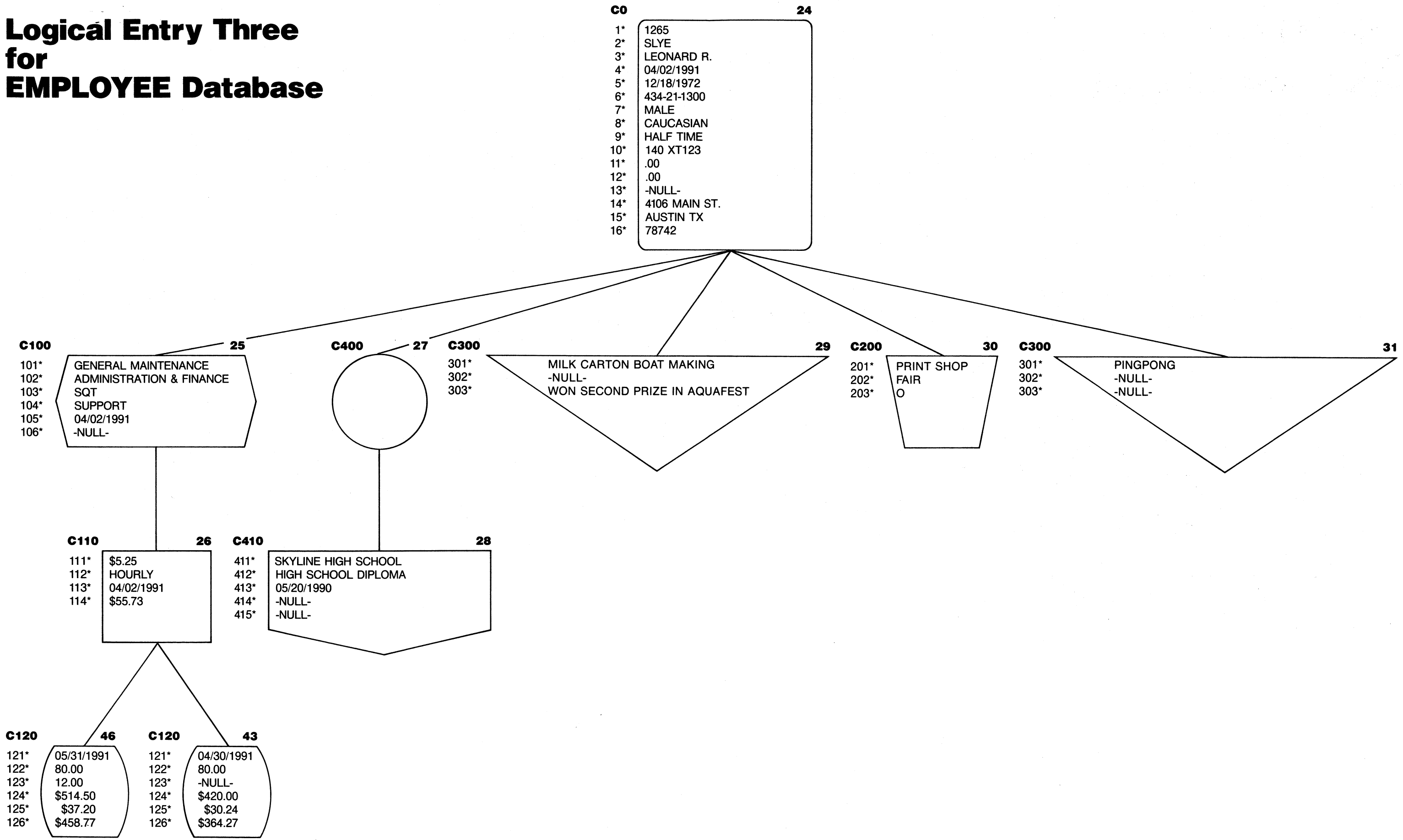
Logical Entry One for EMPLOYEE Database



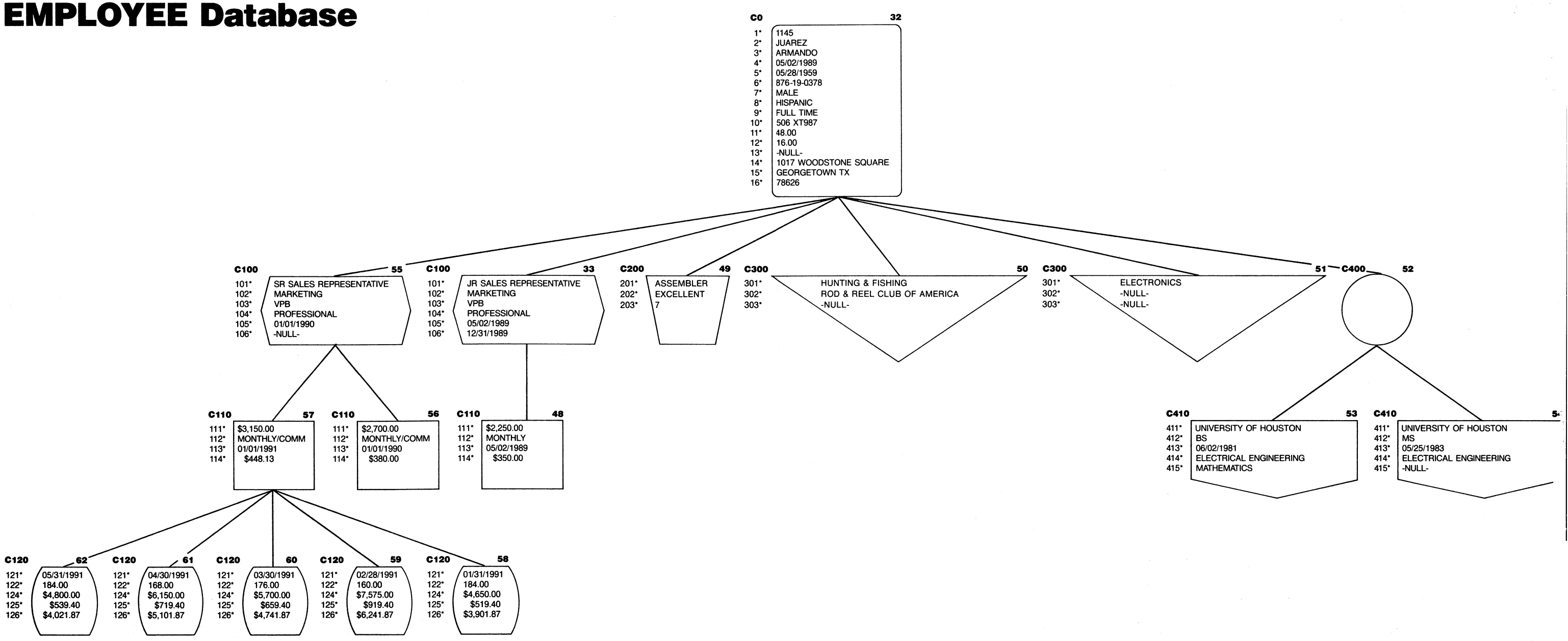
Logical Entry Two for EMPLOYEE Database



Logical Entry Three for EMPLOYEE Database



Logical Entry Four for EMPLOYEE Database



Your Turn

If you have comments or suggestions about SYSTEM 2000 software or *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*, please send them to us on a photocopy of this page.

Please return the photocopy to the Publications Division (for comments about this book) or the Technical Support Division (for suggestions about the software) at SAS Institute Inc., P.O. Box 200075, Austin, TX 78720-0075.