SAS Publishing

SYSTEM 2000®
DEFINE Language

Version 12
First Edition

The Power to Know®

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

SYSTEM 2000® DEFINE Language, Version 12, First Edition

ISBN 1-55544-179-3

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

1st printing, August 1991

Contents

USAGE

REFERENCE

APPENDICES

Illustrations

Using This Book

Purpose

This book provides both usage and reference information for the DEFINE 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 design and define 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 and SYSTEM 2000 terminology. Characteristics common to all SYSTEM 2000 databases are introduced, along with concepts of logical organization and the types of data you can store.

Chapter 1, "Introduction"

Chapter 2, "SYSTEM 2000 Concepts and Terminology"

Chapter 3, "Creating a Database Definition"

Chapter 4, "Displaying a Database Definition"

Chapter 5, "Modifying a Database Definition"

Chapter 6, "Controlling the Processing of a DEFINE Session"

Chapter 7, "Guidelines for Using the DEFINE Language"

Reference The Reference part describes the syntax of DEFINE commands in detail and provides examples.

Chapter 8, "Changing Components: CHANGE"

Chapter 9, "Deleting Components: DELETE"

Chapter 10, "Controlling Execution: ENABLE/DISABLE EXECUTION"

Chapter 11, "Function Definition"

Chapter 12, "Mapping a Definition: MAP"

Chapter 13, "Renumbering a Definition: RENUMBER"

Chapter 14, "Schema Item Definition"

Chapter 15, "Schema Record Definition"

Chapter 16, "Stopping If Errors: STOP AFTER SCAN IF ERRORS OCCUR"

Chapter 17, "String Definition"

Appendices The appendices provide information on various special areas. Among the "Supplementary Topics" discussed in Appendix E are processing large definitions and the Gregorian calendar.

Appendix A, "Collating Sequence and Character Set"

Appendix B, "Special Characters"

Appendix C, "Reserved and Restricted Words and Characters"

Appendix D, "Reserved File Names"

Appendix E, "Supplementary Topics"

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 in this book 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, to emphasize important information, and for comments in sample code.

`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, a vertical bar 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.

ƀ — indicates a significant blank in syntax or output. Generally, spaces indicate blanks, and the ƀ 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. Comments appear in italics and within parentheses.

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 an alternate Command File or Data File is used. Therefore, for example, you cannot have uppercase C3 referring to component number 3 and lowercase c3 referring to component name c3; the software will uppercase c3 and then recognize it as a component number.

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 22 and 12-1. This means that usage information for the MAP command is located on page 22 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 QUEST Language and System-Wide Commands, Version 12, First Edition* (order #A55011) provides concepts and reference material for the SYSTEM 2000 QUEST language and for system-wide commands. For example, this book documents retrieval commands, such as LIST and PRINT, the where-clause, the ordering-clause, update commands, such as INSERT TREE and CHANGE value, and so on.

- *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 a 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 DEFINE language of SYSTEM 2000 software for Version 12. This information is intended for users who have previous experience with the DEFINE language.

Increased Database Size

With Version 12, you can have up to 10,000 database components per database. The previous maximum was 1,000. Note that you will need to calculate the number of buffers required according to the number of components you specify for your database. See **Buffers Required in a DEFINE Session** on page E-3.

In addition, you can specify component numbers from 1 to 9999. The previous range was 1 to 4095.

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 an alternate Command File or Data File is used.

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 be recognized as component number 3.

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

Usage

Chapter 1

Introduction

The DEFINE language for SYSTEM 2000 software consists of commands that allow you to define a SYSTEM 2000 database and modify that definition as well. The language includes easy-to-learn command syntax and a complete set of diagnostic messages.

Types of DEFINE Commands

The DEFINE language consists of three types of commands:

- commands for defining individual database components, which include schema records, schema items, strings, and functions
- commands for modifying an existing database definition
- commands for controlling the processing of DEFINE commands.

Using the DEFINE Language

You can use the DEFINE language at any time in a SYSTEM 2000 session. That is,

- to create a new database, you would issue the CONTROL language NEW DATA BASE IS command, which automatically gives you the DEFINE processor
- to modify an existing database definition, you would issue the CONTROL language DATA BASE NAME IS command with the EXCLUSIVE option and then call the DEFINE processor with the system-wide DEFINE command.

To end a DEFINE session, you would issue the DEFINE language MAP command. The MAP command executes the DEFINE commands you have issued and places the definitions in the database definition. This process is called *mapping a definition.* After mapping the definition, you are given the QUEST processor.

However, before you create a database definition, you must be familiar with some SYSTEM 2000 concepts and terms, which are provided in the following chapter. Also, be sure 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.

Chapter 2

SYSTEM 2000® Concepts and Terminology

INTRODUCTION

SYSTEM 2000 software is a data management system that allows you to organize related data based on a *hierarchical database structure*. In a hierarchical database, you store and access data according to a specific rank or order. That is, associated data are grouped and then ranked based on relationships among the groups of data.

Before you actually create a database, you devise a plan in which you organize the data according to the types of data and how you and others want to use the data.

SYSTEM 2000 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. You view the data logically, but they are stored physically.

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 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. That is, all data records pertaining to a single employee make up a single logical entry. 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 on a form often imply relationships among the data within the sections. These relationships lend structure and meaning to the data. In the same way, the relative position of the data records in a logical entry reflect the structure of the data.

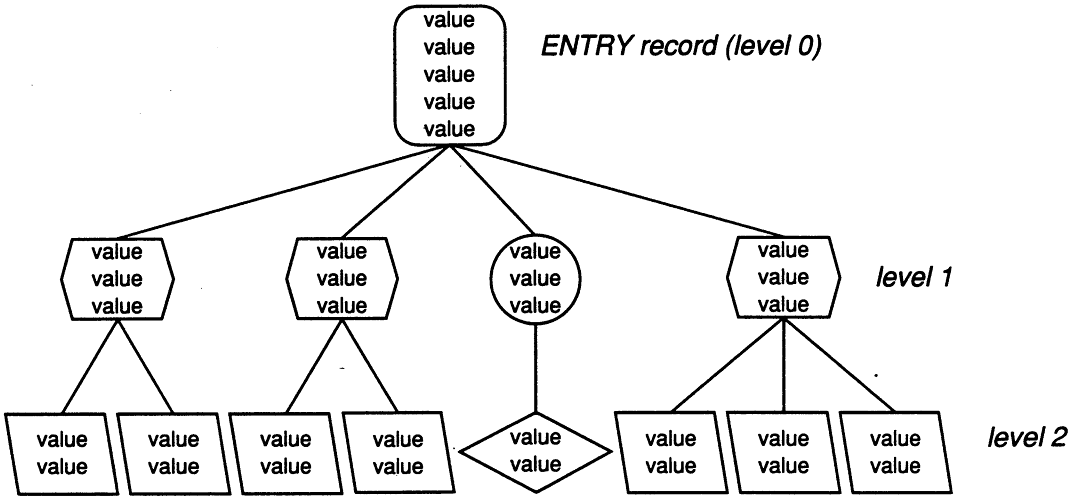
Data records are arranged to reflect the hierarchical structure. That is, each record is placed at a certain level and is related to one or more records in the level immediately below. However, a record can be related to only one record at the next higher level. The highest level is level 0, the next level lower is level 1, and so on.

Hierarchical structures are represented by graphs called trees. Logical entries, therefore, are represented by graphs called *data trees.* In a SYSTEM 2000 database, data trees are shown branching downward. The data record at the top of the tree (that is, the root) is called the *ENTRY record* (level 0).

Data records are represented in a data tree by geometric figures. That is, different data records are represented by different shapes. Hierarchical relationships among those data records are represented by lines connecting the geometric figures.

Illus. 2.1 on page 5 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; for example, the data tree shown in Illus. 2.1 has two kinds of records at level 1.

Illus. 2.1 Logical Entry Containing Data Records



The Database Schema

Continuing 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 a database are filled-in copies of a master entry -- the *database schema*.

Just as the blank form represents a logical view of employee data, the database schema represents a logical view of its data. The schema serves as a pattern to interpret and to create logical entries for a database. A schema consists of schema records and schema items.

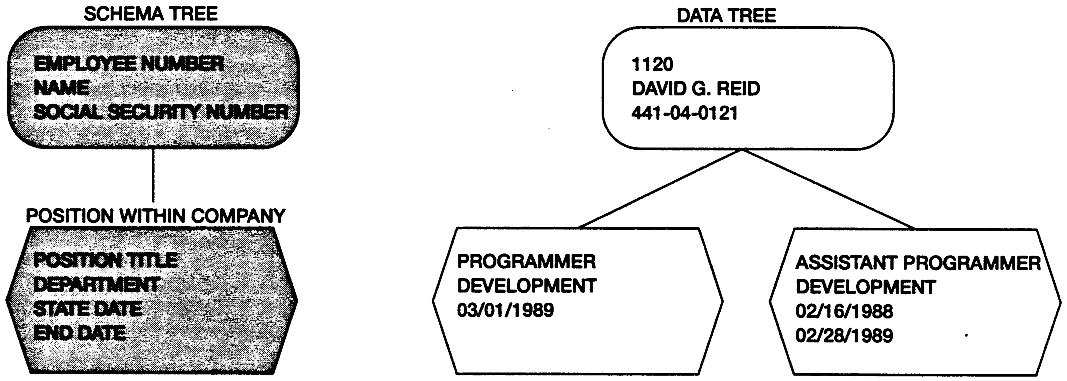
Each value in a data tree corresponds to a *schema item*. A schema item names and defines the characteristics of a group of values. For example, the schema item EMPLOYEE NUMBER (INTEGER 9999) refers to those integers that are used as employee identification numbers. The four 9s indicate that each employee number can contain up to four digits. A schema item belongs to one and only one *schema record*. A schema record is a set of schema items that are treated as a unit. For example, the schema record POSITION WITHIN COMPANY consists of six schema items: POSITION TITLE, DEPARTMENT, MANAGER, POSITION TYPE, START DATE, and END DATE.

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 characteristics that a value must fulfill in order to belong to a given type of data record
- specifies the relationships among records.

In Illus. 2.2 on page 6, the relationship between schema records and data records is shown by displaying a schema tree and a data tree that fit the definition given in the database schema.

Illus. 2.2 Relationship between the Schema Tree and a Data Tree



This illustration shows that in a schema tree there is one distinct geometric figure for each schema record. But in a data tree, the same geometric figure can occur one or more times, or not at all. A schema record is defined only once, but it can be used as many times as needed to create new data records.

A data record is an occurrence of data corresponding to a schema record. Data records that are occurrences of the same schema record are said to be *similar* and are represented by similar geometric figures. Otherwise, they are said to be *dissimilar*.

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

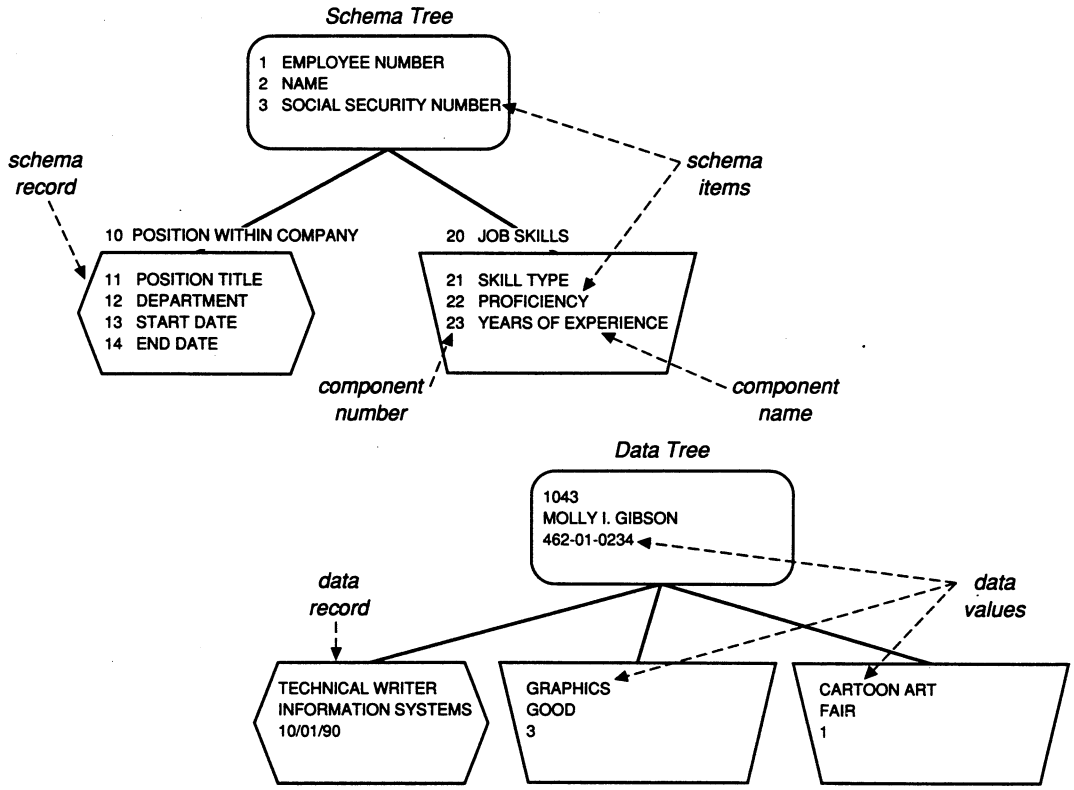
Schema items and schema records are the database *components*. Each component is identified by both a *component number* and a *component name*, as shown below. (A component number can be referred to as a C-number, for example, C101.)

```
      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 within a database to avoid ambiguity.

Illus. 2.3 on page 7 presents some of the previously introduced terms that are referred to with schema and data trees.

Illus. 2.3 Schema and Data Tree Terms



Relationships in a Schema Tree and a 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.

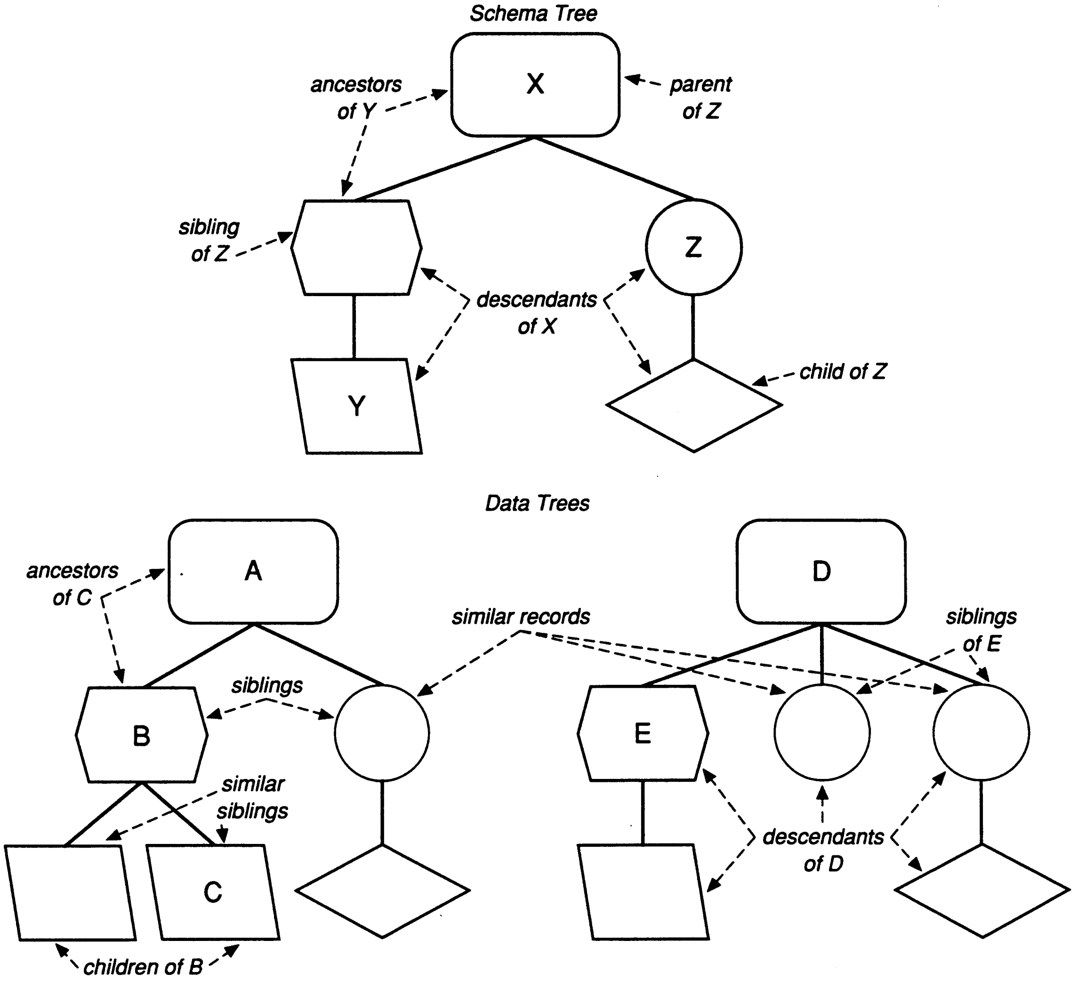
The record immediately above a given record is its *parent*. A record can have only one parent, and the only orphan is the top record, which is called the *ENTRY record.*

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 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.

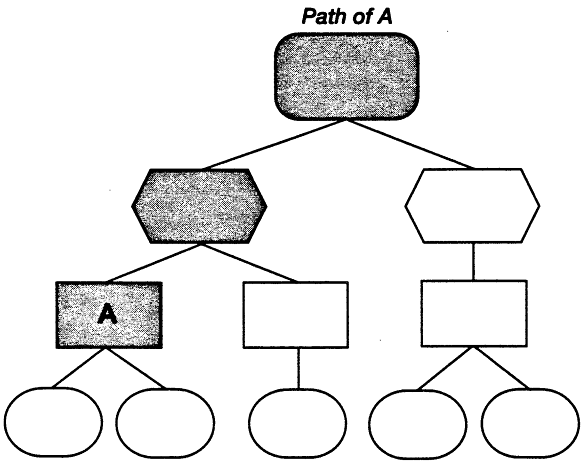
Illus. 2.4 below shows the hierarchical relationships in schema and data trees.

Illus. 2.4 Hierarchical Relationships in Schema and Data Trees



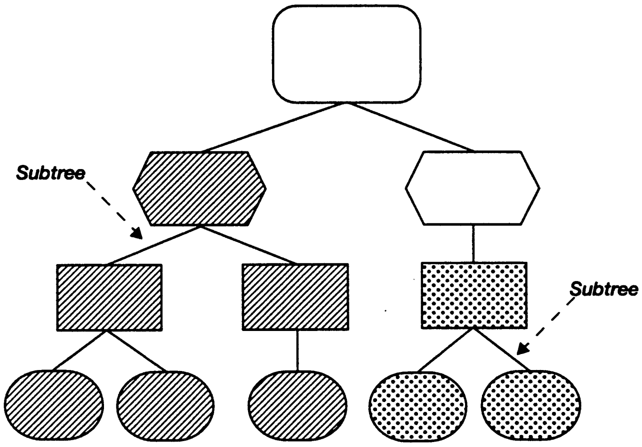
The *path* of a record is the record and all its *ancestors*. While a *family* consists of the record, its ancestors, and its descendants. Illus. 2.5 below highlights a path and a family.

Illus. 2.5 Path and Family



A *subtree* consists of a record and its descendant records, as shown below in Illus. 2.6.

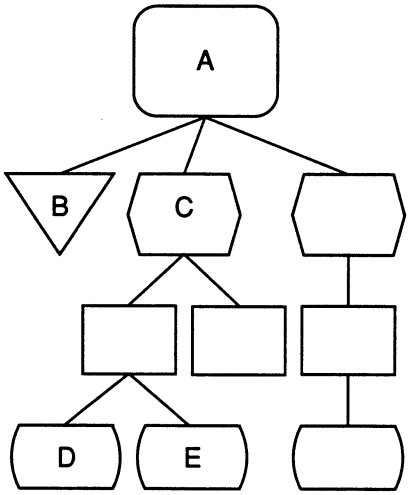
Illus. 2.6 Subtrees



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 when they are occurrences of disjoint schema records, not when they are on different paths.

For example, in Illus. 2.7 below, data records A, C, D, and E are related. Also, data records A and B are related, and data records D and E are related. Data records B and C, however, are disjoint.

Illus. 2.7 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* or *nulls*. For example, on the foldout at the end of this manual, component 13 in the third logical entry of the EMPLOYEE database is a null item, which means that the employee 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 (a null record) must be present in the database since a parent record must exist in the database for every record except the ENTRY record. The fourth logical entry of the EMPLOYEE database has a null ENTRY record.

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.

THE PHYSICAL VIEW OF A SYSTEM 2000 DATABASE

The previous sections present the concepts necessary to plan a database, that is, how to communicate a logical data organization to SYSTEM 2000 software. This section presents, in brief, the structure for physically storing your data, which consists of seven database tables.

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

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

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

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

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

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

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

The following discussion focuses on two of the seven database tables: the Distinct Values Table and the Multiple Occurrence Table. These tables make up the database *index*.

The Index

In a traditional personnel file, folders are usually arranged either alphabetically by employee name or numerically by employee identification number. To access any other information, you must examine each folder, for example, to find employees that know a certain programming language.

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 go through every folder to find certain information; you consult the index instead.

For a SYSTEM 2000 database, when you define a schema item (the database component that stores values), you specify whether the software is to create an index of those values. You specify the indexing status of each item by using the keywords KEY or NON-KEY. That is, the values for a key item are indexed, and the values for a non-key item are not indexed, although they can be searched sequentially.

A key item has an associated index of every distinct value that occurs for the item. This index allows the software to identify and locate quickly and efficiently those records having a specific value or range of values for that item.

In addition to specifying a key status, the CONTROL language CREATE INDEX command also allows you to create an index of values for one or more non-key items. This method is discussed later. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about the CREATE INDEX command.)

You can also create an index for a non-key item by using the DEFINE language to change the item to a key item.

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 Record Number				
78610	3	7			
78620	2	4	6	9	11
78623	1	10	14		
78625	5	8	10	12	

Adding a new employee with a ZIP code of 78620 would require that a new column be added to the index. Since there is no space, the index would have to be extended. To allow for additional column space, you can specify padding for an item at the time it is defined. Then, blank columns would exist at the time the index is created.

On the other hand, if the new employee has a ZIP code of 78630, a new row would have to be added to the index. Again, this index would need to be extended to accommodate the new entry. To allow for additional row space, you can issue the QUEST language ENABLE VALUES PADDING command, which creates blank rows. For additional information on the ENABLE VALUES PADDING command, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

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

<table>
<tr><td>Values</td><td colspan="6">Data Record Number</td></tr>
<tr><td>78610</td><td>3</td><td>7</td><td></td><td></td><td></td><td rowspan="4">This area is padding for multiple occurrences specified in schema item definition.</td></tr>
<tr><td>78620</td><td>2</td><td>4</td><td>6</td><td>9</td><td>11</td></tr>
<tr><td>78623</td><td>1</td><td>10</td><td>14</td><td></td><td></td></tr>
<tr><td>78625</td><td>5</td><td>8</td><td>10</td><td>12</td><td></td></tr>
<tr><td colspan="6">This area is padding for distinct values specified with ENABLE VALUES PADDING.</td><td></td></tr>
</table>

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 one key item (GENDER) that does.

SOCIAL SECURITY NUMBER	
Values	Data Record 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 Record Number					
FEMALE	1	5	9	10		
MALE	2	3	4	6	7	8

Since each employee has a unique social security number, the values for SOCIAL SECURITY NUMBER can occur only once, so multiple occurrence padding is not needed. However, 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 record number" where a value can be found. Actually, the index points to entries in the Hierarchical Table. These entries correspond one-to-one to entries in the Data Table where a data record containing a given value is found.

Chapter 3

Creating a Database Definition

INTRODUCTION

Before you can store and access data in a SYSTEM 2000 database, a database definition must be created. That is, you must

- assign a master password
- assign a name to the database
- define the database components.

After the database definition is created, you can load data into the database.

The following sections provide an overview of how you create a database definition.

ASSIGNING THE DATABASE PASSWORD AND NAME

At the beginning of a SYSTEM 2000 session, the software automatically gives you the CONTROL processor, and the first SYSTEM 2000 command you issue is the USER command, which is a CONTROL language command. The USER command sets the master password for a new database. (It also verifies a password for an existing database to be opened.)

The next command you issue is the NEW DATA BASE IS command, which is also a CONTROL language command. The NEW DATA BASE IS command assigns a name to the new database and automatically calls the DEFINE processor. Note that the NEW DATA BASE IS command automatically gives you exclusive use of the database, which is required by the DEFINE processor.

Examples of a USER command and a NEW DATA BASE IS command that create a new database are as follows:

```
USER, LOAN:
NEW DATA BASE IS BANK:
```

For information on the USER command and the NEW DATA BASE IS command, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

DEFINING THE DATABASE COMPONENTS

After you have specified a master password and assigned a database name, you can issue DEFINE commands to define the database *components* that make up the database definition. There are four types of database components:

- schema records
- schema items
- stored strings
- stored functions.

Schema records and schema items make up the database schema, while all four components make up the definition. Each component has a unique number and name assigned to it. The component number and name are referred to as *labels*, which you use to access data in retrieval and update commands.

The maximum number of components allowed for a database definition is 10,000, with the default being 430. A database can have any number of components between one and the limit set for the database. (You can specify a maximum number of components with the NEW DATA BASE IS command.) The total number of schema records, schema items, and stored strings and functions can be up to the specified or default limit. See Appendix E for the number of buffers required in a DEFINE session.

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). Therefore, for example, you cannot have uppercase C3 referring to component number 3 and lowercase c3 referring to component name c3; the software will uppercase c3 and then recognize it as a component number. To disable uppercase translation, set the execution parameter OPT043 to yes. For information on the OPT043 execution parameter, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

Defining Schema Records

A schema record groups associated schema items. That is, different schema records store different groups of data. To define a schema record, you specify a unique component number and name for the record and identify the component as a record. If the record is at level 2 or below (that is, level 3, level 4, and so on), you must also specify a parent record.

For example, the following commands define a level 1 schema record and a level 2 schema record, respectively:

```
100* POSITION WITHIN COMPANY (RECORD):

110* SALARY WITHIN POSITION (RECORD IN 100):
```

Defining Schema Items

A schema item names and defines the characteristics of a group of values. That is, a schema item has a name, a type, and a picture (length). To define a schema item, you specify

- a unique component number and a name, which labels the schema item
- 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
- the item type and picture, which determine how the data are used and displayed
- the schema record to which the schema item belongs.

For example, the following command defines schema item EMPLOYEE NUMBER (C1). C1 will store numbers with a picture of 9999, and the item is at level 0 because there is no schema record specification. Since non-key is not specified, the item defaults to key, which means that there will be an index of values.

```
1* EMPLOYEE NUMBER (INTEGER NUMBER 9999):
```

The following command defines schema item POSITION TITLE (C101). The item is defined as a non-key item (no indexing), and it will store character values up to ten characters. The item is also defined as belonging to schema record C100, which is a level 1 record.

```
101* POSITION TITLE (NON-KEY CHAR X(10) IN 100):
```

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 you can use the values. For example, values consisting exclusively of digits can be stored in a manner suitable for computation. SYSTEM 2000 software offers a variety of character and numeric item types and a date item type. The following item types are available:

- CHARACTER items store character data with trailing, leading, and extra internal blanks removed. For example, JOHNƀƀƀSMITH is stored and displayed as JOHNƀSMITH.
- DATE items store calendar dates in a fixed format, for example, 09/24/1991.

- DECIMAL items store decimal values.

- DOUBLE items store double word (double-precision) floating point values.

- INTEGER items store integer values.

- MONEY items also store decimal values, but when displayed, they include a floating dollar sign at the left and CR at the right (if negative).

- REAL (or FLOAT) items store single-precision (full-word) floating point values.

- TEXT items also store character data; however, blanks are not removed. For example, ƀƀJOHNƀƀƀSMITHƀƀƀ is stored and displayed as ƀƀJOHNƀƀƀSMITHƀƀƀ.

- UNDEFINED items store binary bit-string data containing any of the 256 EBCDIC characters.

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.

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. You can also use the numeral 9 followed by the number of places in parentheses, for example, 9(4) is equivalent to 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 if the specified picture is at least X(4). The picture is indicated by repetitions of the letter X. You can also use the letter X, followed by the number of places in parentheses. For example, X(3) is equivalent to XXX.

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

In addition to schema records and items, a database definition can also include stored strings and functions as components. Strings and functions provide both convenience and security.

- *Strings* allow you to store commands, parts of commands, or a series of commands. For example, strings provide a way for users who are not familiar with the QUEST language to access data.

- *Functions* allow you to store arithmetic expressions and to specify a numeric type for the display of output. For example, a stored function can provide the current age of each employee in a company.

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

After strings and functions are stored in a definition, you can execute them by their component name or number with the QUEST, QUEUE, or REPORT languages, which will in turn invoke the stored syntax.

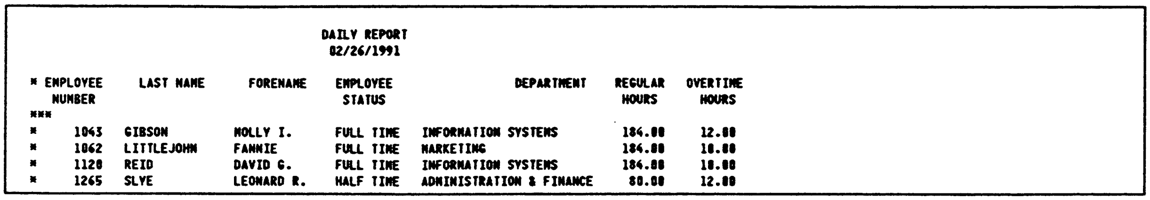
Strings Suppose you prepare a daily report for the personnel manager, using the following QUEST language LIST command:

```
LIST/TITLE D(40) DAILY REPORT ,EMPLOYEE+ NUMBER,R(12)
   LAST NAME   ,R(11) FORENAME ,EMPLOYEE +
   STATUS,R(24)      DEPARTMENT       ,REGULAR+ HOURS,
   R(7)OVERTIM+ HOURS,B(1)E/C1,C2,C3,C9,C102,C122,C123,
   ORDERED BY LAST NAME
   WHERE ((C123 GT .01 AT 1) AT 1) AT 1:
```

The length and complexity of this LIST command make it an ideal candidate for storage in a string, thereby avoiding coding errors every time the report is needed and allowing any user to access the desired data.

You could store this LIST command, for example, in a string named DAILY REPORT with component number C1001. The report could then be produced at any time by issuing either of the following commands:

```
*C1001*

*DAILY REPORT*
```



```
                                      DAILY REPORT
                                       02/26/1991

* EMPLOYEE     LAST NAME      FORENAME    EMPLOYEE                  DEPARTMENT   REGULAR   OVERTIME
  NUMBER                                   STATUS                                 HOURS     HOURS
***
*      1043  GIBSON         MOLLY I.     FULL TIME   INFORMATION SYSTEMS          184.00    12.00
*      1062  LITTLEJOHN     FANNIE       FULL TIME   MARKETING                    184.00    10.00
*      1120  REID           DAVID G.     FULL TIME   INFORMATION SYSTEMS          184.00    10.00
*      1265  SLYE           LEONARD R.   HALF TIME   ADMINISTRATION & FINANCE      80.00    12.00
```

This report is arranged in alphabetical order by last name. You may need the same report to be arranged in alphabetical order by department or in numerical order by employee number. This need for order flexibility can be met by making the sort item a *parameter* of the string, thereby making the string a *parametric string.*

A parametric string contains one or more parameters (such as *1*, shown below) for which actual values are supplied when the string is invoked. The string definition would be as follows:

```
1001* DAILY REPORT (STRING (LIST/TITLE D(40)+DAILY REPORT+,
          EMPLOYEE+ NUMBER,
          R(12) LAST NAME  ,R(11) FORENAME , EMPLOYEE + STATUS
          R(24)        DEPARTMENT       ,REGULAR+ HOURS,
          R(7)OVERTIM+  HOURS,B(1)E/C1, C2, C3, C9, C102, C122
          C123, ORDERED BY *1* WHERE ((C123 GT .01 AT 1) AT 1:)):
```

When you invoke the string, you must supply a schema item to replace the *1* in the string definition. Therefore, the following command produces the same report as the original string:

```
*DAILY REPORT (LAST NAME)
```

Functions The following QUEST language LIST command displays the last name and age of each employee in the EMPLOYEE database. (FTODAY is a system string that supplies the current date.)

```
LIST LAST NAME, ((*FTODAY* - BIRTHDAY)/365.25):
```

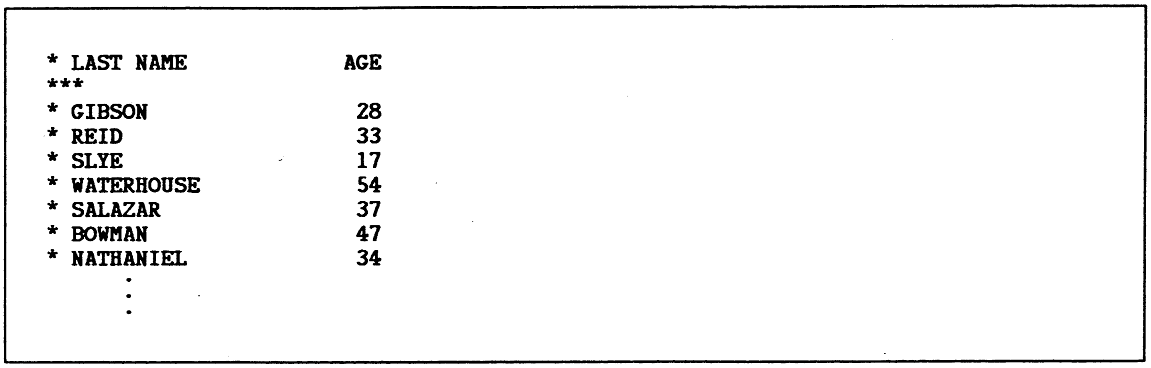
```
* LAST NAME
***
* GIBSON                    28.945
* REID                      33.191
* SLYE                      17.848
* WATERHOUSE                54.962
* SALAZAR                   37.864
* BOWMAN                    47.280
* NATHANIEL                 34.593
      :
      .
```

Notice that the output shows age in fractions of years. To display age in whole numbers and also to provide a heading to the age column, you can store the arithmetic expression used to calculate age as an INTEGER function, as shown next:

```
2001*AGE (INTEGER FUNCTION $ ((*FTODAY* - BIRTHDAY)/365.25)$):
```

Then you can obtain the results you want by issuing the following command:

```
LIST  LAST NAME, *AGE*:
```



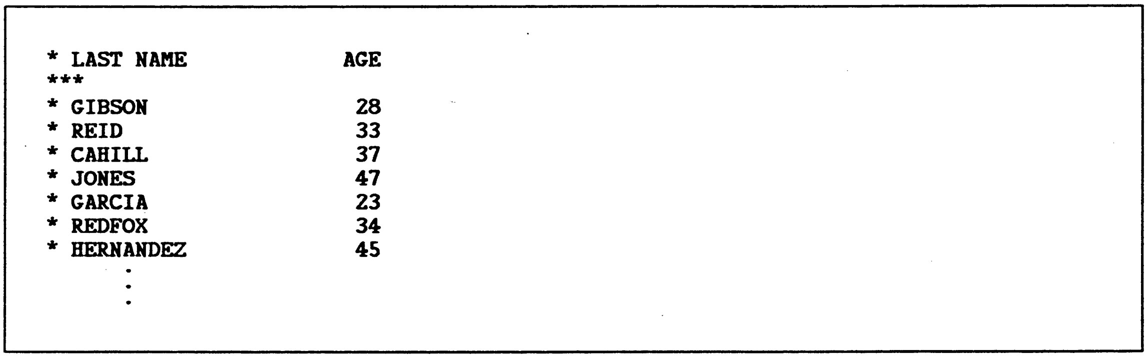
```
* LAST NAME            AGE
***
* GIBSON                28
* REID                  33
* SLYE                  17
* WATERHOUSE            54
* SALAZAR               37
* BOWMAN                47
* NATHANIEL             34
      .
      .
      .
```

Functions can also be parametric, and they can be combined with strings. For example, suppose you define the following string:

```
1002* AGE REPORT (STRING $ LIST LAST NAME, *AGE* WHERE DEPARTMENT
EQ *1*:$):
```

You can then invoke the stored string as shown next to produce the following report:

```
*AGE REPORT (INFORMATION SYSTEMS)
```



```
* LAST NAME            AGE
***
* GIBSON                28
* REID                  33
* CAHILL                37
* JONES                 47
* GARCIA                23
* REDFOX                34
* HERNANDEZ             45
      .
      .
      .
```

The call to the string invokes the AGE function and substitutes INFORMATION SYSTEMS for the parameter *1*. The report displays the last name and age of each employee in the INFORMATION SYSTEMS department.

Note: When a string or function invokes another string or function, it is referred to as a *nested* string or function. AGE REPORT is a nested string since it invokes the AGE function.

ENDING A DEFINE SESSION AND MAPPING THE DEFINITION

To end a DEFINE session, issue the MAP command. The MAP command processes the DEFINE commands you have submitted and places the results in the database definition. After you issue the MAP command, the software switches to the QUEST processor.

To call the DEFINE processor again, you can issue the DEFINE command. Or to end the SYSTEM 2000 session, you can issue the EXIT command.

Note: If you call another processor or issue the EXIT command prior to issuing the MAP command, the DEFINE commands in the session are cancelled.

EXAMPLE: CREATING A DATABASE

The following example illustrates the general sequence of commands needed to define a new database definition:

```
USER, TELL:
NEW DATA BASE IS BANK:
1* CUSTNAME (CHAR X(20)):
2* CUSTID (CHAR X(7)):
100* ACCOUNTS (RECORD):
101* ACCOUNT NUMBER (INTEGER 9(4) IN 100):
102* ACCOUNT TYPE (CHAR X IN 100):
200* TRANSACTIONS (RECORD IN 100):
201* TRANS TYPE (CHAR X IN 200):
202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
203* TRANS DATE (DATE IN 200):
MAP:
EXIT:
```

Note the following points about these commands:

- At the beginning of a SYSTEM 2000 session, the CONTROL processor is automatically attached, and the first command you issue is the USER command. This command sets the master password for the new database, which is TELL in this example.

- The NEW DATA BASE IS command, a CONTROL language command, assigns the name to a new database, which is BANK in this example, and calls the DEFINE processor.

- The component definition commands assign a unique number and name to each component and specify an item type and picture for each schema item.

- The definitions define two schema records (C100 and C200) and seven items. SYSTEM 2000 software automatically creates the level 0 record (the ENTRY record) for schema items C1 and C2.

- Only the TRANS AMOUNT item is specified as non-key. Since non-key is not specified for the other items, SYSTEM 2000 software defines them as key items, by default, and indexes the values.

- The schema items that belong to a particular schema record are designated with the IN specification. For example, items C101 and C102 belong to record C100; items C201, C202, and C203 belong to record C200. The IN specification also determines which records are subordinate to other records. Here record C200 belongs to record C100.

- If an item does not have an IN specification, it automatically belongs to the level 0 record (the ENTRY record). Since an IN specification is not included for items C1 and C2, they are at level 0.

- The MAP command is required when you define a database or when you modify a definition. MAP processes the commands and writes the definition to the database files.

- To end a SYSTEM 2000 session, you issue the EXIT command, which is a system-wide command.

PROCESSING ORDER FOR DEFINITIONS

SYSTEM 2000 software accepts component definitions in any order, as long as a schema record is defined before its member items and descendant records. The order in which definitions are processed affects the structure of the schema that results. That is,

- the order of record definitions establishes the order of output from a DESCRIBE command for sibling schema records. See Chapter 4, "Displaying a Database Definition" for an explanation of the DESCRIBE command and its output. Also, see the foldout at the end of this manual for complete DESCRIBE command output for the EMPLOYEE database.

- the order of item definitions establishes the order of schema items within a schema record.

Component numbers are used by SYSTEM 2000 software only to identify the components specified in a command; after identification, SYSTEM 2000 software uses an internally assigned tag for each component. Component numbers are a convenient label, usually shorter than the component name. Also, the numbers can aid your memory in recalling logical groups of components. For example, if C100 is a record label, then items C101, C102, and C103 can be visually associated as members of C100 records. You might want to reserve a range of component numbers for strings and functions.

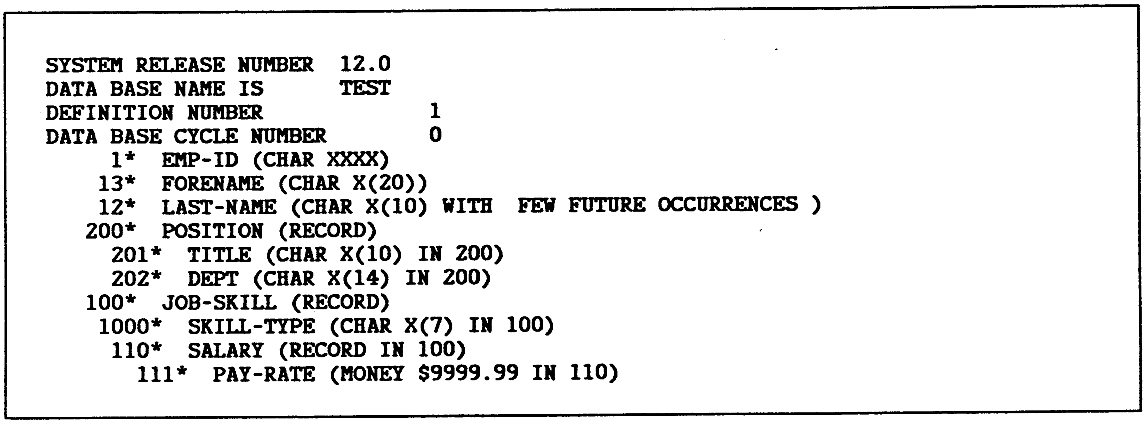
It is good practice to enter component definitions in logical order. That is, enter all ENTRY record (level 0) schema item definitions first, followed by level 1 schema records and associated items, and so on. Entering your definition in logical order can save processing time for a large definition. If the new definition is not entered in logical order, considerable processing is required to rearrange the existing components.

This order is referred to as *DESCRIBE command output order*. In DESCRIBE command output order, items are grouped under the appropriate record. However, items within a record remain in the order in which they were entered.

You can define strings and functions in any order. The DEFINE processor separates strings from functions and places functions before strings, with each set occurring in order as defined.

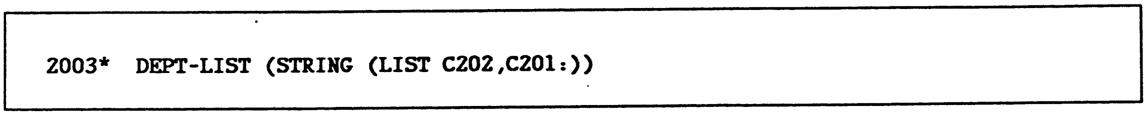
To illustrate the reordering of components performed by the DEFINE processor, the following definitions are not assigned in logical groups. Note that the input order, rather than the component number, establishes DESCRIBE command output order.

```
USER, TEST:
NEW DATA BASE IS TEST:
1* EMP-ID (CHAR X(4)):
13* FORENAME(CHAR X(20)):
200* POSITION (SCHEMA RECORD):
100* JOB-SKILL (SCHEMA RECORD):
201* TITLE (CHAR X(10) IN 200):
1000* SKILL-TYPE (CHAR X(7) IN 100):
202* DEPT (CHAR X(14) IN 200):
2003* DEPT-LIST (STRING?LIST C202,C201:?):
12* LAST-NAME (CHAR X(10) WITH FEW FUTURE OCCURRENCES):
110*SALARY (SCHEMA RECORD IN 100):
111*PAY-RATE (MONEY 9999.99 IN 110):
MAP:
DESCRIBE:
```



```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      TEST
DEFINITION NUMBER         1
DATA BASE CYCLE NUMBER    0
     1*  EMP-ID (CHAR XXXX)
    13*  FORENAME (CHAR X(20))
    12*  LAST-NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
   200*  POSITION (RECORD)
     201*  TITLE (CHAR X(10) IN 200)
     202*  DEPT (CHAR X(14) IN 200)
   100*  JOB-SKILL (RECORD)
    1000*  SKILL-TYPE (CHAR X(7) IN 100)
     110*  SALARY (RECORD IN 100)
       111*  PAY-RATE (MONEY $9999.99 IN 110)
```

```
DESCRIBE STRINGS:
```



```
2003*  DEPT-LIST (STRING (LIST C202,C201:))
```

Chapter 4

Displaying a Database Definition

INTRODUCTION

After you have defined a database definition or modified one, you can use the QUEST language DESCRIBE command to display all or part of a database definition to check it.

The following information explains how to use the DESCRIBE command. For complete information on the DESCRIBE command, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

THE DESCRIBE COMMAND

Using the DESCRIBE command, 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 specified security for controlling 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 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, for example, that each component description ends with a command terminator.

  If you set the Report File as an alternate file, you can use DESCRIBE /DEFINE/ output written to an alternate Report File as a Command File. By doing so, you can quickly create new database definitions that are similar to existing definitions.

  When using the /DEFINE/ option with Collect File items, the Collect File items are automatically in the level 0 record and are numbered sequentially starting with 1. This allows you to create a new database with the Collect File items.

- 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.

Output from a DESCRIBE command begins with header information, as shown in Illus. 4.1 on page 27. (See the corresponding reference numbers.)

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 or non-key status, record membership, and padding. (Items are key unless NON-KEY is displayed in the item definition.)

The database schema shown in Illus. 4.1 on page 27 is an excerpt from the EMPLOYEE database. See the foldout at the end of this manual for the complete schema.

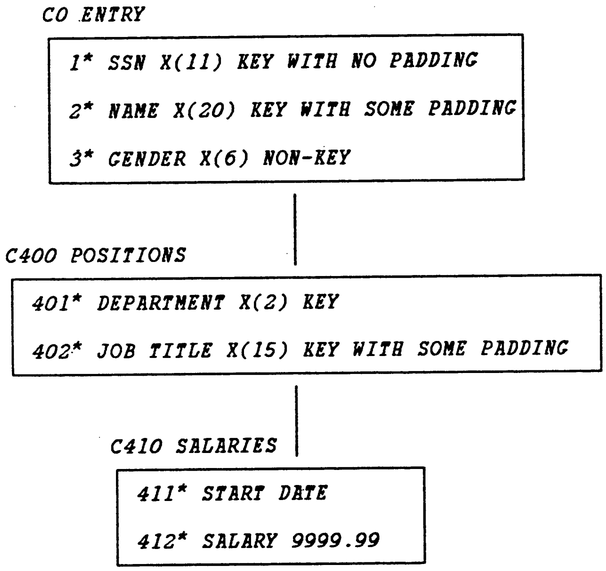
Illus. 4.1 DESCRIBE Command Output

```
SYSTEM RELEASE NUMBER       12.0  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)

       120*  MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)

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

     130*  ADDITIONAL INFORMATION (RECORD IN 100)

       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 )
```

EXAMPLE: CREATING AND DISPLAYING A NEW DEFINITION

The following example illustrates planning a database, the commands to define the definition, and the results of issuing DESCRIBE commands to check the definition:

1. Suppose you design the following plan for a test database:



2. You then issue the following DEFINE commands to create the database definition. Note that the DEFINE commands are syntactically correct, but they do not define the database definition exactly as presented in the above plan.

```
USER,ONE:
NEW DATA BASE IS TEST:
1* SSN (CHAR X(11)):
3* GENDER (NON-KEY CHAR X(6)):
400* POSITIONS (RECORD):
401* DEPARTMENT (CHAR X(2) IN 400):
402* JOB TITLE (CHAR X(15) IN 400):
410* SALARIES (RECORD):
411* START DATE (DATE IN 410):
412* SALARY (DEC NUMBER 9999.9 IN 410):
1000* SERVICE TIME (FUNCTION/(END DATE - START DATE)/):
MAP:
```

3. After the definition is mapped, which gives you access to the QUEST processor, you then can issue DESCRIBE commands to check the definition.

```
DESCRIBE:
DESCRIBE FUNCTIONS:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS       TEST
DEFINITION NUMBER             1
DATA BASE CYCLE NUMBER        0
     1*  SSN (CHAR X(11))
     3*  GENDER (NON-KEY CHAR X(6))
   400*  POSITIONS (RECORD)
     401*  DEPARTMENT (CHAR XX IN 400)
     402*  JOB TITLE (CHAR X(15) IN 400)
   410*  SALARIES (RECORD)
     411*  START DATE (DATE IN 410)
     412*  SALARY (DECIMAL NUMBER 9999.9 IN 410)

1000*  SERVICE TIME (DECIMAL FUNCTION ((END DATE - START DATE)))
```

4. As mentioned, the DEFINE commands entered in Step 2 are syntactically correct. However, the DESCRIBE output reveals discrepancies between the displayed definition and the original database plan. The discrepancies are as follows:

   - An item (C2 * NAME) is missing from the ENTRY record.
   - The component name for C402 is not spelled correctly.
   - No padding exists for C402.
   - Record C410 has C0 as its parent, instead of C400 as specified in the plan.
   - Item C412 has the wrong picture.
   - String C1000 contains a reference to a component that does not exist.

To fix the definition, you can issue DEFINE commands, which are discussed in Chapter 5, "Modifying a Database Definition."

CONSIDERATIONS FOR USING THE DESCRIBE COMMAND

You can use DESCRIBE output to check for the following criteria:

- Logical structure (record definitions) considerations:
  - Does each record have the intended component number and name?
  - Are all records in the database design included?
  - Does each record have the intended parent?
  - Does each record have the intended sibling(s)?
  - Does each record have all the items that should be included in it and only those items?

  - Are items listed in the intended order?

- Item definitions considerations:

  - Does each item have the intended component number and name?

  - Does each item have the intended key/non-key status?

  - Is the padding specification correct for each key item?

  - Are the item type and picture for each item accurate?

- String or function definitions considerations:

  - Do string or function definitions contain references to components with changed component numbers or names?

  - Do strings or functions contain references to deleted components?

  - Do any strings containing commands, such as REPORT or LIST, need to change because of item type or picture changes?

- PLEX program considerations:

  - Do all PLEX programs use the current logical structure?

  - Do PLEX subschema records use pictures and types compatible with new item definitions?

  - Do any PLEX component names need to be changed?

  - Do PLEX subschema records contain any deleted components, or do they need to have any new components added? New components need to be added only if new logic in the program is to use the new components.

Chapter 5

Modifying a Database Definition

INTRODUCTION

The DEFINE language can accomplish many kinds of definition modifications. You can

- add new items, records, strings, and functions at any time in the life of a database by defining new components in the same manner as defining a new definition.
- delete any component at any time with the DELETE command. In some cases, you can accomplish your modifications by deleting a component and defining a new one with the desired characteristics. In other cases, you can use the CHANGE command to achieve the desired change.
- change one or more parts of an individual component with the CHANGE command. For example, you can change characteristics of individual components, such as name, number, picture, and padding specification.
- renumber the components with the RENUMBER command.

Before a database is loaded with values (populated), modifications to its definition are easily processed since such modifications affect only the Master Record and Definition Table; these changes are quite inexpensive. Once a database is populated, it is simple and still inexpensive to change the Definition Table for component names, numbers, and padding. Also, you can add, delete, and modify strings and functions at any time, because these changes are reflected only in the Definition Table.

Some modifications, however, affect other database tables as well (the index and Data Table), which might automatically *restructure* the database. Restructuring a database means that a definition change causes SYSTEM 2000 software to automatically unload all data and then reload the data back into the new definition. As a result, all database tables are rebuilt to accommodate the changes.

For example, if you change the component name of an item, no restructuring takes place. On the other hand, since the database schema contains the hierarchical structure, changes to record relationships and item characteristics cause restructuring. In addition, restructuring occurs from several types of schema modifications, such as modifying a picture and adding or deleting schema records or items. (Note that for deleted components, restructuring removes the stored data.)

Although restructuring is done automatically by SYSTEM 2000 software, you must provide adequate scratch file space. For more information on restructuring, see **Automatic Restructuring** on page E-5.

GENERAL CONSIDERATIONS FOR MODIFYING A DEFINITION

The following considerations are relevant to definition modification:

- When a database definition needs modifying, it is more efficient to issue all changes that cause restructuring in one DEFINE session.

- You can assign deleted component numbers and names to components defined after the deletion, even in the same DEFINE session.

- You can change component labels at any time without causing restructuring.

- When you change a component label, all references to the old label in strings, REPORT language, or PLEX programs become unusable. Use output from DESCRIBE STRINGS and DESCRIBE FUNCTIONS commands to check component labels in stored strings or functions.

- You can add and delete components at any time. Component numbers do not affect position in the schema. A new component always appears in DESCRIBE output as the last record under its parent record or as the last item within a record. When you add sibling records, they appear in input order as the last records for the parent. When you add several items, they appear in input order as the last items for the record.

- You can add or delete schema items at any time. When you add a schema item, restructuring stores a corresponding null item in each data record. When you delete an item, all values for the item are removed from the database.

- When you are changing a schema item, whether a data record exists for the schema record containing that item has a direct influence on what types of changes are allowed and when restructuring is necessary. A data record is said to exist even though it may have been removed from the logical structure. The removed data record becomes part of reusable space and stays there until reused or until a reload or restructure process occurs. For information on when restructuring occurs, see **Automatic Restructuring** on page E-5.

- Existing values are never changed, even if the picture or item type is modified or if restructuring occurs for any reason.

- If a schema record is deleted from the database, all of its descendant schema records are also deleted, and the values for all of the data records are removed. See **Alternate Methods for Changing a Schema** on page E-7 for a discussion of alternate methods of preserving data for future use.

- The definition number is always incremented by one when a MAP command completes its processing. The database cycle number is reset to 0 or 1 during a restructuring process. It is set to 0 if no data qualify (from a specified where-clause), and it is set to 1 if any data records and values remain after restructuring. If no restructuring takes place, the database cycle number is unchanged.

- When you modify a definition that causes restructuring, the Update Log is suspended. Therefore, consider the following procedure:

  1. Before modifying a database definition that has the Update Log activated, issue a KEEP command

  2. To copy the database before modifying its definition, issue a SAVE command.

  3. After modifying the definition, reactivate the Update Log by issuing a SAVE command

  In addition, definition changes are never recorded on the Update Log. If there is no restructuring, there is no logged indication that the definition was changed. You can use the KEEP and SAVE commands to create a break in the Update Log, at which point definition changes can be reintroduced during the recovery process.

  The SAVE, KEEP, and RESTORE commands, as well as the Update Log, are discussed in detail in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

- The DEFINE processor requires exclusive use of a database. Therefore, you can do definition modifications in a production environment. You can release exclusive use immediately after completing a DEFINE session, and all users can then call any Self-Contained Facility language to retrieve and update the database concurrently. However, you would not normally modify the definition of a production database in such a casual manner. Typically, you would test the proposed definition changes in a single-user environment before introducing it into a Multi-User environment.

  For example, to test schema changes, consider applying the changes to a test database consisting of a subset of the production database. This type of testing lets you analyze the effects of schema modification and consider possible alternatives, as well as set up and test pertinent strings and functions. When you test, be sure to use an adequate subset of the database (for example, with values for each item) so that any potential restructuring does indeed occur. Restructuring in a test situation is desirable so that you can plan for data unloading or adequate scratch file space to handle restructuring of the production database.

  Consider issuing the CONTROL language SAVE command to save a copy of a database before modifying a definition in a way that will cause restructuring. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for information on the SAVE command.

Illus. 5.1 on page 34 lists the types of definition changes you can do and when they are allowed.

Illus. 5.1 Types of Definition Changes

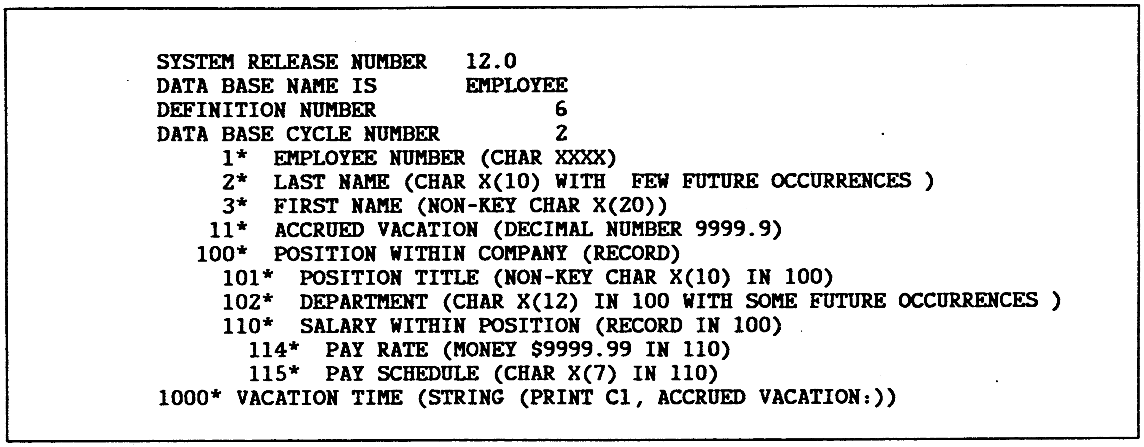
Type of Change	When Allowed
Adding a component	Always.
Deleting a component	Always.
Changing a component label	Always.
Changing key/non-key status	Always.
Changing padding specification	Always.
Changing item types:	
character to character	Always.
character to numeric	If data record does not exist.
character to date	If data record does not exist.
numeric to numeric	Always.
numeric to character	If data record does not exist.
numeric to date	If data record does not exist.
date to character	If data record does not exist.
date to numeric	If data record does not exist or if numeric item picture is 9(7).
Changing picture:	
enlarging character item pictures	Always
reducing character item pictures	If data record does not exist or if reduced picture is X(4) or larger.
moving decimal point in numeric picture	Always
enlarging or reducing numeric picture	If data record does not exist.

Note that a data record is considered to exist even though it may have been removed from the logical structure.

EXAMPLE: MODIFYING A DEFINITION

The following commands and output illustrate modifying a database definition by deleting components, adding new components, and changing component descriptions: This example uses an excerpt of the EMPLOYEE database.

1. The original database has the following definition as displayed by a DESCRIBE command:



```
SYSTEM RELEASE NUMBER    12.0
DATA BASE NAME IS        EMPLOYEE
DEFINITION NUMBER               6
DATA BASE CYCLE NUMBER          2
     1*  EMPLOYEE NUMBER (CHAR XXXX)
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
     3*  FIRST NAME (NON-KEY CHAR X(20))
    11*  ACCRUED VACATION (DECIMAL NUMBER 9999.9)
   100*  POSITION WITHIN COMPANY (RECORD)
     101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
     102*  DEPARTMENT (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCES )
     110*  SALARY WITHIN POSITION (RECORD IN 100)
       114*  PAY RATE (MONEY $9999.99 IN 110)
       115*  PAY SCHEDULE (CHAR X(7) IN 110)
1000* VACATION TIME (STRING (PRINT C1, ACCRUED VACATION:))
```

2. The following DEFINE commands modify the definition:

```
DEFINE:
DELETE COMPONENT 100:
100* JOB SKILLS (RECORD):
110* SKILL TYPE (CHAR X(12) IN 100):
111* CURRENT DEDUCTION (NON-KEY MONEY 999.99 IN 110):
CHANGE 1 TO 1* EMPLOYEE-ID (CHAR X(3)):
CHANGE 3 TO 3*FIRST NAME (NON-KEY TEXT X(10)):
CHANGE 11 TO 11*VACATION TIME (DEC NUMBER 999.99):
MAP:
```

Note the results of these DEFINE commands:

- The DEFINE command calls the DEFINE processor.
- The DELETE command deletes record C100, its descendant record C110, and associated schema items.
- The new component definitions reuse component numbers 100 and 110.
- The new item C111 becomes the last item in its record, which is the new C100.
- Changing the picture for C1 is rejected because a data record exists for that component, and the new picture is not large enough to allow values that exceed the picture. Because the picture is not acceptable, the component number is not changed either.

- Changing C3 from one character item type to another is accepted. The decrease in picture is also accepted.

- Changing the component name and picture for C11 is acceptable, but the change leaves an obsolete reference to the old name in string C1000.

- The MAP command processes the DEFINE commands, maps the new definition, and calls the QUEST processor. The definition number is increased, and the restructuring process resets the database cycle number to 1.

3. After the previous DEFINE commands are executed, the database definition is as follows:

```
DESCRIBE:
DESCRIBE STRINGS:
```

```
SYSTEM RELEASE NUMBER   12.0
DATA BASE NAME IS       EMPLOYEE
DEFINITION NUMBER              7
DATA BASE CYCLE NUMBER         1
     1*  EMPLOYEE NUMBER (CHAR XXXX)
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
     3*  FIRST NAME (NON-KEY TEXT X(10))
    11*  VACATION TIME (DECIMAL NUMBER 999.99)
   100*  JOB SKILLS (RECORD)
     101*  SKILL TYPE (CHAR X(12) IN 100)
     111*  CURRENT DEDUCTION (NON-KEY MONEY $999.99 IN 110)
1000* VACATION TIME (STRING (PRINT C1, ACCRUED VACATION:))
```

Chapter 6

Controlling the Processing of a DEFINE Session

When you issue a DEFINE command, SYSTEM 2000 software scans the command for errors. If the command contains no error, it is accepted; however, the database definition is not changed until you issue the MAP command. When the software finds an error, it displays the appropriate error message. When you issue the MAP command, thus ending a DEFINE session, the definition is mapped into the database Definition Table.

Three DEFINE commands interact with the MAP command by either allowing or preventing mapping of error-free commands, thus giving you control over the definition process.

ENABLE EXECUTION
allows the MAP command to proceed as described above and maps all correct commands. ENABLE EXECUTION is in effect at the start of a SYSTEM 2000 session.

DISABLE EXECUTION
prevents mapping of any DEFINE language commands, regardless of whether they are acceptable. After using DISABLE EXECUTION to check for errors during the DEFINE session, you must regain access to the DEFINE processor, issue the ENABLE EXECUTION command, and resubmit the DEFINE commands.

STOP AFTER SCAN IF ERRORS OCCUR
prevents mapping of all DEFINE commands if any errors occur and if ENABLE EXECUTION is in effect. You can then edit the commands in error and resubmit them. If the software encounters no errors, mapping occurs.

Another method of controlling the processing of the MAP command is to append a where-clause and an optional ordering-clause. When changes to the schema of a populated database restructure the database tables, SYSTEM 2000 software retains the logical entries that meet the qualification criteria specified in the where-clause and arranges the logical entries as specified in the ordering-clause.

An Update Log is available to record updates to values and the data structure. However, the Update Log does not record definition changes. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of using the Update Log in regard to definition modification.

Chapter 7

Using the DEFINE Language

INTRODUCTION

This chapter contains an assortment of topics that will help you use the DEFINE language.

CHANGING PROCESSORS

The processor lookup capability provides you some flexibility in changing from one processor to another. If processor lookup is activated (with the system-wide LOOKUP IS ON command) and you issue a command that is not valid for the current processor, SYSTEM 2000 software tries to switch you to the appropriate processor. For more information on the LOOKUP IS command, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

You can also issue the system-wide commands QUEST, CONTROL, and DEFINE to call the respective processor, so that you can issue commands specific to that processor. Additionally, the CONTROL language NEW DATA BASE IS command automatically calls the DEFINE processor.

ISSUING DEFINE COMMANDS

The following general guidelines apply to DEFINE 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.

- You can continue commands from one line to the next.

- You can enter more than one command on a line.

- Each command must end with a command terminator. The default command terminator is the colon.

The following example illustrates the different ways you can issue DEFINE commands:

```
1* EMP-ID (CHAR X(4)):  13* FORENAME(CHAR X(20)):
200* POSITION (SR):          100* JOB-SKILL (SR):
12* LAST-NAME (CHAR X(10) WITH FEW FUTURE OCCURRENCES):  110*
SALARY(SR IN 100):
```

CHECKLIST FOR DEFINE SYNTAX

Problems in using the DEFINE language often result from syntax errors, mistakes in component labels, effects of previous commands such as DISABLE EXECUTION, or effects of previous commands that were not executed, such as a record definition.

The following guidelines will help you avoid errors:

- Every component name and number must be unique within a database.

- You cannot use reserved words in component names, even when preceded and followed by hyphens or other special symbols. For example, -OR- is not acceptable. (Reserved words are listed in Appendix C.)

- Do not include C with a component number; that is, specify 100, not C100.

- You must specify a system separator between the number and name in the component definition.

- There must be a right parenthesis for every left parenthesis.

- Each command must end with a command terminator. Because SYSTEM 2000 software has no way of knowing where you intended the command to end, this error may go undetected until later in the command stream.

- Each definition must include the keyword RECORD, STRING, or FUNCTION, or an item type specification.

- A specified picture cannot be more than 250 characters for character item types and fifteen for numeric item types.

- Be careful with defaults when defining components. A problem might result later from the omission of a specification, which produced a definition that you did not want. For example, you might erroneously define a schema item as a member of the C0 record by not including an IN specification, or you might have a definition that uses a default picture that you did not intend, or you might have a key item because you did not specify NON-KEY.

- Any unmapped commands are lost when you leave the DEFINE processor. For the commands to be executed, you must call the DEFINE processor again, re-enter the commands, and then issue the MAP command.

PROCESSING ENVIRONMENTS

You can use the DEFINE language in either batch or interactive mode. Also, you can use the DEFINE language in either a single-user or Multi-User execution environment.

In a single-user environment, you are working with your own copy of SYSTEM 2000 software. You usually have exclusive access to the database, although the single-user environment can be configured so that all users can query the database.

In a Multi-User environment, many users can access a database at the same time, with queries and updates being handled by the Multi-User software simultaneously for all databases being accessed. Remember that the DEFINE language requires exclusive use of a database.

For information on a single-user or Multi-User environment, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

SYSTEM 2000 FILES

SYSTEM 2000 software uses several kinds of files. For single-user jobs, you must allocate all files in your Job Control Language (JCL) or with a CLIST or EXEC. In a Multi-User environment, files are allocated when the Multi-User software is initialized, or you can let the software allocate most of the work files dynamically. For more information on file allocation, see the NEW DATA BASE IS command in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*. You can also allocate your database files dynamically using the CONTROL language ALLOC and FREE commands. For information on these commands, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

Database Files

Database files must be allocated for each database. Each database has six required files, plus two optional files used for recovery purposes; these files are maintained by the software. The database files are as follows:

- File 1 - Master Record and Definition Table
- File 2 - Distinct Values Table
- File 3 - Extended Field Table
- File 4 - Multiple Occurrence Table
- File 5 - Hierarchical Table
- File 6 - Data Table
- File 7 - Update Log (optional)
- File 8 - Rollback Log (optional)

The DDnames for the database files must contain the first seven characters of the database name, plus a digit from 1 to 8. If the database name is less than seven characters, you must use Xs in the DDname to make it seven characters long. For example, if you were to name a database AUTO, the DDname for File 1 would be AUTOXXX1.

Sort Files

Sort files sort and merge lists of addresses. SYSTEM 2000 software sorts addresses of data records rather than the actual records. Also, the software sorts only the addresses of records selected by a command, not the entire database.

Scratch Files

Scratch files hold information temporarily, such as between processing one part of a SYSTEM 2000 command and another. Scratch files also hold input data in various formats during the load processes. SYSTEM 2000 software keeps all scratch files in operating system data sets called *scratch pads*.

User Files

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

- The Command File is an input file containing commands you submit to SYSTEM 2000 software.

  Note: One technique for defining a database is to keep the definition in an alternate Command File, change the file as needed during planning, and use the file to create a test database definition when appropriate.

- The Data File is an input file containing data used by the QUEST language LOAD command.

- The Report File is an output file containing retrieval results.

- The Message File is an output file containing echoes of commands and messages sent to you by SYSTEM 2000 software.

The two input files (the Command File and the Data File) are named INPUT by default. INPUT identifies the device (such as your terminal) from which the software reads either commands or data. The two output files (the Report File and the Message File) are named OUTPUT by default. OUTPUT identifies the device (such as your terminal or printer) on which the software writes messages and retrieval results. The software will allocate these four files dynamically if the DDnames (S2KCOMD and S2KMSG) are not present in the JCL.

You can specify alternate user files by issuing the system-wide commands COMMAND FILE IS, DATA FILE IS, MESSAGE FILE IS, and REPORT FILE IS, which are discussed in *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*.

Optional Files

A few special SYSTEM 2000 capabilities require optional files. Saving a database requires a Savefile, update logging requires a Keepfile, and PLEX programs require a Locate File. Also, you can specify an S2KPARMS file that contains execution parameters.

For information on the Savefile and Keepfile, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*, for information on the Locate File, see the *SYSTEM 2000 PLEX Manual, Version 12, First Edition*, and for information on the S2KPARMS file, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

SECURITY CONSIDERATIONS

SYSTEM 2000 software provides security based on passwords. You can associate three types of passwords with a SYSTEM 2000 database:

- a master password, which is required
- secondary passwords
- a DBA password.

The master password holder has unqualified access to the database. This is the password under which a database is created, and the only password that can modify a database definition. The master password holder can also assign multiple secondary passwords with authorities assigned to individual database components and a DBA password with authorities assigned to individual SYSTEM 2000 commands.

For more information on SYSTEM 2000 passwords, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

MESSAGES

During a SYSTEM 2000 session, messages appear in your S2KMSG file as necessary in response to your commands. There are three general types of messages:

Informative — appear during normal processing, informing you of procedures undertaken and completed by the software, such as a restructuring of the database. Informative messages are not fatal to the job and are never destructive to your database. The condition code is 0.

Warning — tell you that the software has executed an acceptable command that may have undesirable consequences. The condition code is 4.

Error — notify you that a command contains an error. The DEFINE processor messages are numbered with the 100 series numbers. The condition code is 8 for nonfatal errors and 12 for fatal errors.

For more information about messages that can appear during a DEFINE session, see *SYSTEM 2000 Messages and Codes, Version 12, First Edition*. Also, see **Checklist for DEFINE Syntax** on page 40.

Reference

Changing Components: CHANGE

Deleting Components: DELETE

Controlling Execution: ENABLE/DISABLE EXECUTION

Function Definition

Mapping a Definition: MAP

Renumbering a Definition: RENUMBER

Schema Item Definition

Schema Record Definition

Stopping if Errors: STOP AFTER SCAN IF ERRORS OCCUR

String Definition

Chapter 8

Changing Components: CHANGE

The CHANGE command changes any part of an individual component definition except record membership and parent record. That is, you can use the CHANGE command to change a component's number, name, and description (or any combination). The new definition replaces the previous definition. Note that for the C0 component (the ENTRY record), you can change only the component name.

All constraints on defining new components apply to the CHANGE command. For example, you can specify padding for key items only, you cannot use reserved words in component names, and so on. For other constraints on defining new components, see Chapter 11, "Function Definition," Chapter 14, "Schema Item Definition," Chapter 15, "Schema Record Definition," and Chapter 17, "String Definition."

If you want to change any part of a component description, you must include all specifications or SYSTEM 2000 software uses defaults. That is, non-default specifications do not carry over from the initial definition. For example, if you include an item description (that is, item type, picture, key or non-key status, or padding), you must include the entire item description. If no specification is included for the picture, the key or non-key status, or the padding, the software assigns defaults. In this situation, defaults can cause unwanted or even unacceptable changes to the overall definition.

If there are no data records for the schema record (and no reusable space for the record) in the database, any change in item type or picture is acceptable, except you cannot change item types REAL, DOUBLE, or UNDEFINED to any other item type or any other item type to REAL, DOUBLE, or UNDEFINED. If the database contains values, there are additional restrictions on changes in item type and picture. For those restrictions, see **Changes in Item Type** on page 8-3 and **Changes in Picture** on page 8-4.

You cannot use the CHANGE command in the following situations:

- to change parent-child record relationships or the record membership of an item.

- to change a schema item to a schema record, function, or string. Also, you cannot change a record to a string, function, or item, or a string or function to a record or item. However, you can change a string to a function or a function to a string.

Format

```
CHANGE old-number TO new-number :
```

```
CHANGE old-number TO |new-number * new-name :
                     |old-number
```

```
CHANGE old-number TO |new-number * |new-name (new-description) :
                     |old-number   |old-name
```

old-number is the number of an existing component, without the C.

new-number is a new number for a component, without the C. The number can be from 1 to 9999, but it cannot duplicate any existing number. Changing the component number of a schema record automatically changes the number in any IN specification for member items and descendant records.

new-name is a new name for a component, using 1 to 250 characters. It cannot duplicate any existing name, and it cannot contain reserved words or characters. For a list of reserved and restricted words and characters, see Appendix C. Also, the component name cannot contain the current system separator. The default system separator is an asterisk (*), which you can change with the system-wide SEPARATOR IS command.

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). To disable uppercase translation, set the execution parameter OPT043 to YES.

old-name is the name of an existing component. If you do not specify an old name, only the component number is changed. Note that an old name is allowed only if it corresponds to an old number; for example, when the number and/or description is changed but the name is not.

new-description defines a new description for a component. You must specify a new name or the old name when specifying a new description. You cannot make changes to the IN specification. If you do not include a new description, only the component name and/or number is changed, and the existing description is retained.

Abbreviation: CHANGE = CH

Changes in Item Type

If there are no data records for a schema record (and no reusable space for the record) in the database, any change in item type is acceptable, except as noted below.

When data records exist for the schema record containing the item whose type you need to change, you can use the CHANGE command. The new item type must use the same storage format, that is, packed decimal or character, as the old one. You can change most numeric item types to other numeric item types, most character item types to other character item types, and a DATE item type to numeric item types. However, you cannot change a REAL, DOUBLE, or UNDEFINED item to any other item type, and you cannot change any other item type to REAL, DOUBLE, or UNDEFINED. Compatible item types are as follows:

Item Type	Compatible Types
CHARACTER	TEXT
TEXT	CHARACTER
INTEGER	MONEY, DECIMAL
MONEY	DECIMAL, INTEGER
DECIMAL	INTEGER, MONEY
DATE	INTEGER, DECIMAL, MONEY

Changes in item type can have significant consequences for data already in a database. Data are not edited to conform to the new definition; that is, the stored data do not change.

For example, a TEXT item retains all blanks. If you change a TEXT item to a CHARACTER item, previously stored values are retained with blanks as entered. Therefore, changing the item type to CHARACTER has implications for where-clause processing. The extraneous blanks stored as part of the values will need to be specified in qualification criteria, or else they will prevent a match. As a CHARACTER value having multiple blanks, it is not accessible as qualification criteria unless delimiters are used in the where-clause condition. Otherwise, extraneous blanks are edited out in the where-clause processing, preventing a match with stored values.

If you would rather remove the extraneous blanks, you can unload the data, make the item type change, and re-enter the data using one of the alternate methods discussed in **Alternate Methods for Changing a Schema** on page E-7. In this case, the where-clause delimiter does not have to be used, because re-entering the data removes the extraneous blanks. (For a discussion on the where-clause delimiter, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*.)

Dates are stored as seven-digit, packed-decimal (four bytes) values: YYYMMDD where YYY is (year-1500). If you change a date to a numeric type (other than REAL and DOUBLE), previously stored dates are no longer displayed as month/day/year. Dates are then displayed as a series of seven digits with the decimal placed as specified for the new picture. Therefore, if you change a DATE item to a numeric item, the picture must have seven digits.

Changes in Picture

Numeric items You can change the picture for a numeric item (DECIMAL, MONEY, and INTEGER) with the following exception: if a data record for the item has existed since the last reload or restructuring, the total number of digits in the picture cannot change. For example, changing 9(3).9(2) to 9(4).9(1) is acceptable, but changing 9(3).9(2) to 9(3).9(1) is not.

If you change a date to a numeric item type, the overall picture must have seven digits. The data stored for the items do not change. For example, a value of 123.60 for a DECIMAL item type with a picture of 9(4).9(2) becomes a value of 12360 if changed to INTEGER 9(6).

Previously stored values are not changed by picture changes or by restructuring, so they may have different meanings than when the data were loaded. For example, if the picture for a DECIMAL item is changed from 9(3).9(2) to 9(4).9(1), then the implied decimal point moves one place to the right, and a value such as 123.45 would be treated as 1234.5.

In the previous example, you could use the QUEST processor to multiply the values by 10, for example,

```
CHANGE C111 = (C111*10) where-clause
```

You can also use the process of deleting and redefining an item to adjust database values. For information on the process, see **Alternate Methods for Changing a Schema** on page E-7.

Character items The picture for a CHARACTER or TEXT item can be made smaller or larger, with the following exception: if a data record for the item has existed since the last reload or restructuring, the picture can become smaller only if the new picture is equal to or greater than X(4). This is because a minimum of four characters is needed to take care of any values that exceed the picture, and no mechanism exists for decreasing a picture that is smaller.

Considerations

- CHANGE commands do not affect the order of DESCRIBE command output.
- Some definition changes, such as changing a non-key item to key or a key item to non-key, cause automatic restructuring of the database. For a discussion of when restructuring occurs, see **Automatic Restructuring** on page E-5.
- You can use the CREATE/REMOVE INDEX command to change key or non-key status without restructuring the database. For a comparison of the CHANGE command and the CREATE/REMOVE INDEX command, see **Automatic Restructuring** on page E-5. For information on the CREATE/REMOVE INDEX command, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*
- Changing a component name or number may require that you also change references to that name or number within stored strings, stored functions, and PLEX programs.

Examples

These examples illustrate the CHANGE command.

Example 1

The following CHANGE commands change component numbers and names but retain the original component descriptions:

```
CHANGE 400 TO 500:           (specifies a new number only)

CHANGE 712 to 712*XYZ:       (specifies a new name only)

CHANGE 700 TO 800*ABC:       (specifies a new number and a new name)

CHANGE 0 TO 0*EMPLOYEE:      (changes level 0 ENTRY to EMPLOYEE)
```

Example 2

The following command changes a component's number, name, and description:

```
CH 201 TO 210*SKILLS (CHAR X(24) IN 200 WITH MANY FUTURE OCCURRENCES):
```

Example 3

The following command changes a function's definition:

```
CHANGE 2000 TO 2000* FUNA (FUNCTION $((COUNT C600)/*1*)$):
```

Example 4

Using the EMPLOYEE database, the following CHANGE commands are invalid:

```
CHANGE 0 TO 100:
(You cannot change a CO number.)

CHANGE 200 TO 200*DEF (SR IN 100):
(You cannot change an IN specification.)

CHANGE 132 to 132*COMMENTS (SR IN 130):
(You cannot change a schema item to a schema record.)

CHANGE 303 to 303*COMMENT TEXT:
(You cannot request a duplicate name - C132.)
```

Chapter 9

Deleting Components: DELETE

The DELETE command removes components from a database definition. You can specify individual components or a range of components to be deleted. If the database has been populated, associated data records and values are also removed.

You can use keywords to limit the range of components to be deleted. Each keyword designates one or more types of components, as explained in the following syntax.

Deleting a schema record removes all member items and descendant records from the database. These items and records are removed even though they are not listed in the command. However, references to deleted components within string definitions are not removed.

Unlike other DEFINE commands, a DELETE command can be partially executed despite a syntax error. If there is an error in a component number, components to the left of the error are deleted. See Example 2 in this chapter.

Format

```
DELETE |COMPONENTS          number |[, number] ...     :
       |SCHEMA COMPONENTS          |[THROUGH number]
       |STRINGS
       |FUNCTIONS
```

COMPONENTS specifies schema records, schema items, strings, and functions.

SCHEMA COMPONENTS specifies schema records and schema items.

STRINGS specifies strings only.

FUNCTIONS specifies functions only.

Abbreviations:
COMPONENTS = COMPONENT
FUNCTIONS = FUNCTION
STRINGS = STRING
THROUGH = THRU or TO

number is the number of an existing component, without the C. The component number must identify the components of the type specified by the keyword.

Components (listed individually and separated by commas) can be in any order.

If you specify THROUGH (for example, 600 THROUGH 605), the first component must precede the second component in DESCRIBE command output.

Consideration

Some deletions can result in automatic restructuring of the database by SYSTEM 2000 software. For information on when restructuring occurs, see **Automatic Restructuring** on page E-5.

Examples

These examples illustrate the DELETE command.

Example 1

The following DELETE commands illustrate the command's syntax:

```
DELETE COMPONENTS 410, 408, 412:

DELETE COMPONENT 2 THROUGH 4:

DELETE STRING 1001:

DELETE FUNCTION 2020, 2050:
```

Example 2

In the following example, the second DELETE command has an error in the last specified component number. However, the first two components in the command are accepted and deleted. Notice that descendant records of the deleted record C110 are also deleted.

Note that this example uses an excerpt from the EMPLOYEE database.

```
DESCRIBE:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             4
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)
   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 )
     110*  SALARY WITHIN POSITION (RECORD IN 100)
       111*  PAY RATE (MONEY $9999.99 IN 110)
       120*  MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)
         121*  PAYROLL MONTH (DATE IN 120)
   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)
      .
      .
      .
```

```
DEFINE:
DELETE COMPONENT 2 THROUGH 4:
DELETE COMPONENT 101,110,300:
...........................C
```

```
-115- COMPONENT REQUESTED NOT FOUND -
```

```
MAP:
DESCRIBE:
```

```
-516-  UPDATE LOG SUSPENDED -
-128- 'DEFINE' COMMAND HAS CAUSED DATA BASE RESTRUCTURING -
-242- 19:15:58 - HIERARCHICAL STRUCTURE UPDATING COMPLETE -
-245- 19:15:58 - RECORD UPDATING COMPLETE -
-246- 19:15:58 - BEGIN KEY VALUE SORT -
-247- 19:15:58 - KEY VALUE SORT COMPLETE -
-248- 19:15:58 - BEGIN LONG VALUE SORT -
-249- 19:15:58 - LONG VALUE SORT COMPLETE -
-250- 19:15:58 - LONG VALUE UPDATE COMPLETE -
-251- 19:15:58 - LOADING COMPLETE -
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            5
DATA BASE CYCLE NUMBER       1
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
   100*  POSITION WITHIN COMPANY (RECORD)
     102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
   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)
      .
      .
      .
```

Chapter 10

Controlling Execution: ENABLE/DISABLE EXECUTION

The ENABLE EXECUTION and DISABLE EXECUTION commands allow SYSTEM 2000 software to scan DEFINE commands while either enabling or disabling their execution when a MAP command is issued. The function of the two commands is as follows:

- The ENABLE EXECUTION command, which is set by default at the beginning of a SYSTEM 2000 session, allows mapping to occur for all syntactically correct commands.
- The DISABLE EXECUTION command prevents the mapping of all commands. When the MAP command is issued, all DEFINE commands from the session are discarded regardless of whether they are syntactically correct. A message is displayed, and the software switches to the QUEST processor. You must re-enter the discarded definitions in a new DEFINE session under the control of the ENABLE EXECUTION command.

ENABLE EXECUTION remains in effect until you issue a DISABLE EXECUTION command and vice versa, even if you change processors.

All commands are processed without regard to any of the following edit-control commands:

```
ENABLE EXECUTION:
DISABLE EXECUTION:
STOP AFTER SCAN IF ERRORS OCCUR:
```

The current setting for edit-control is effective when MAP is issued and only then. Using multiple interleaved ENABLE/DISABLE EXECUTION commands has no effect other than the effect of the most recent command when MAP is issued. That is, there is no control over the use and non-use of individual sets of definitions within one DEFINE session.

Format

```
ENABLE EXECUTION :

DISABLE EXECUTION :
```

Consideration

You can issue ENABLE/DISABLE EXECUTION from the QUEST and QUEUE processors as well as from the DEFINE processor. The command affects commands in all three processors, regardless of which processor it is issued from. The command remains set throughout a SYSTEM 2000 session.

Examples

These examples illustrate the ENABLE EXECUTION and DISABLE EXECUTION commands.

Example 1

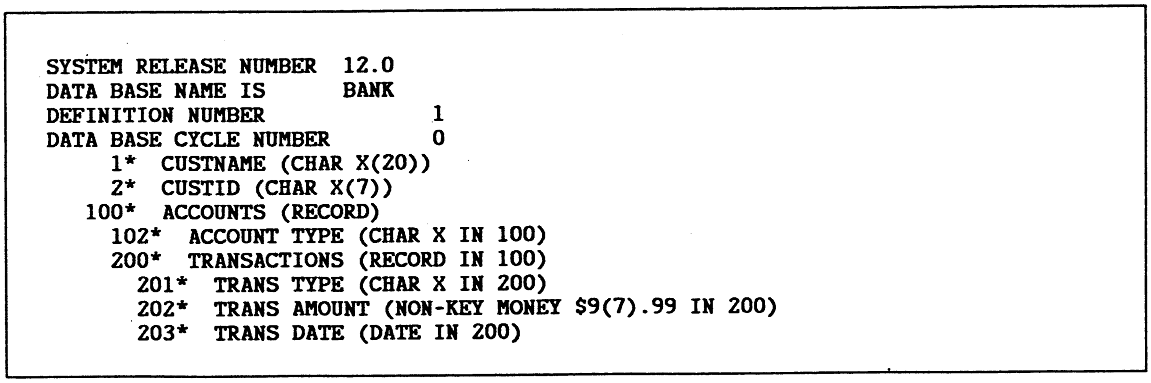
The following example illustrates how ENABLE EXECUTION allows mapping of syntactically correct commands, if the software encounters a command with an error. An arrow points to the DEFINE command with a definition error; a numeric item type's picture must be specified with a 9, not an X.

```
   USER, TELL:
   NEW DATA BASE IS BANK:
   ENABLE EXECUTION:
   1* CUSTNAME (CHAR X(20)):
   2* CUSTID (CHAR X(7)):
   100* ACCOUNTS (RECORD):
→  101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
   102* ACCOUNT TYPE (CHAR X IN 100):
   200* TRANSACTIONS (RECORD IN 100):
   201* TRANS TYPE (CHAR X IN 200):
   202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
   203* TRANS DATE (DATE IN 200):
   MAP:
```

The following messages are displayed after the MAP command is executed:

```
11    NEW DATA BASE IS BANK:
  -558- CREATED....BANK                        0          0   03/04/1991  14:12:10
12    ENABLE EXECUTION:
13    1* CUSTNAME (CHAR X(20)):
14    2* CUSTID (CHAR X(7)):
15    100* ACCOUNTS (RECORD):
16    101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
 .............................C
  -100- SYNTAX ERROR -
17    102* ACCOUNT TYPE (CHAR X IN 100):
18    200* TRANSACTIONS (RECORD IN 100):
19    201* TRANS TYPE (CHAR X IN 200):
20    202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
21    203* TRANS DATE (DATE IN 200):
22    MAP:
```

The resulting definition is as follows:



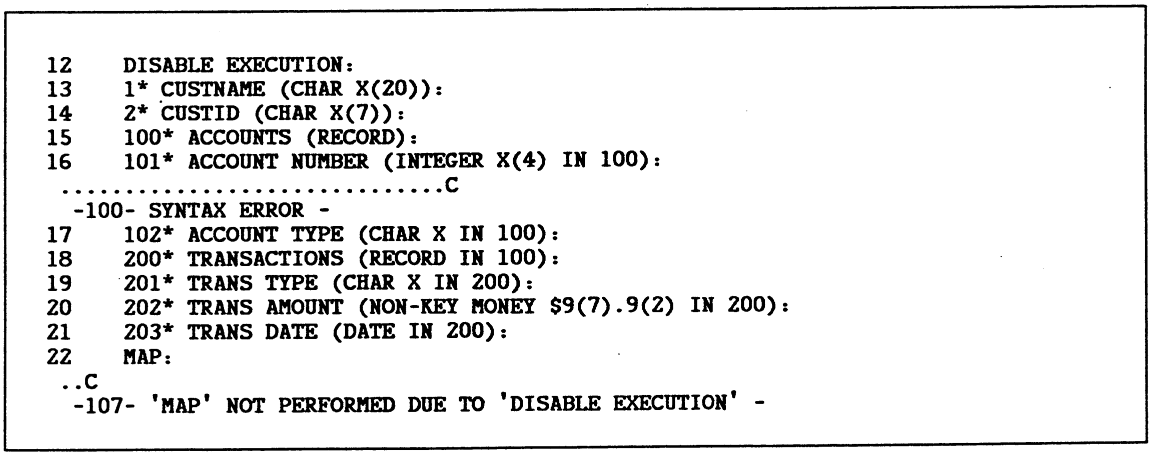
```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      BANK
DEFINITION NUMBER            1
DATA BASE CYCLE NUMBER       0
     1*  CUSTNAME (CHAR X(20))
     2*  CUSTID (CHAR X(7))
   100*  ACCOUNTS (RECORD)
     102*  ACCOUNT TYPE (CHAR X IN 100)
     200*  TRANSACTIONS (RECORD IN 100)
       201*  TRANS TYPE (CHAR X IN 200)
       202*  TRANS AMOUNT (NON-KEY MONEY $9(7).99 IN 200)
       203*  TRANS DATE (DATE IN 200)
```

Example 2

The following example illustrates how DISABLE EXECUTION prevents mapping of all DEFINE commands and then discards the commands if an error is encountered. As in Example 1, an arrow points to the erroneous DEFINE command.

```
  USER, TELL:
  NEW DATA BASE IS BANK:
  ENABLE EXECUTION:
  1* CUSTNAME (CHAR X(20)):
  2* CUSTID (CHAR X(7)):
  100* ACCOUNTS (RECORD):
→ 101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
  102* ACCOUNT TYPE (CHAR X IN 100):
  200* TRANSACTIONS (RECORD IN 100):
  201* TRANS TYPE (CHAR X IN 200):
  202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
  203* TRANS DATE (DATE IN 200):
  MAP:
```

The following messages are displayed after the MAP command is executed:



```
12    DISABLE EXECUTION:
13    1* CUSTNAME (CHAR X(20)):
14    2* CUSTID (CHAR X(7)):
15    100* ACCOUNTS (RECORD):
16    101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
 .............................C
  -100- SYNTAX ERROR -
17    102* ACCOUNT TYPE (CHAR X IN 100):
18    200* TRANSACTIONS (RECORD IN 100):
19    201* TRANS TYPE (CHAR X IN 200):
20    202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
21    203* TRANS DATE (DATE IN 200):
22    MAP:
 ..C
  -107- 'MAP' NOT PERFORMED DUE TO 'DISABLE EXECUTION' -
```

Chapter 11

Function Definition

Functions store arithmetic operations, such as addition, subtraction, multiplication, and division. This type of function is referred to as a *stored function*. By invoking a stored function, you request the processing of the stored syntax without having to repeat the full command syntax.

You invoke a function by including its component name or number in the action-clause of QUEST retrieval and update commands or in a REPORT action-clause. For more information on invoking stored functions, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

Functions provide convenience and security. Because functions are stored in a database definition and requested later by their component numbers or names, retrievals and updates are not prone to coding errors or errors in command construction.

Functions can contain:

- parameters
- other functions or strings
- arithmetic expressions
- system functions (AVG, COUNT, MAX, MIN, SUM, SIGMA)
- system string *FTODAY*.

There are two kinds of functions: functions that are used exactly as defined by issuing their component name or number, and *parametric* functions that have parameters for which values must be provided when the function is issued.

You can specify up to 31 levels of nesting, 31 parameters, any combination of the two yielding a total of 31, or a maximum of 1200 characters of function expansion and/or parameters. Parameters can be supplied within a function definition calling another string or function with parameters.

Functions are subject to the same security measures as other components. A secondary password may have authority to use only a specified set of functions. Also, the password used when a function is invoked must have proper authority for all components referenced in the stored command.

Format

```
number * name (|[DATE]      FUNCTION delimiter (expression) delimiter) :
               |[DECIMAL]
               |[DOUBLE]
               |[INTEGER]
               |[MONEY]
               |[REAL]
```

number is an integer from 1 through 9999 to be the component number of the function. Each component number must be unique within a database. Note that functions are included in the maximum number of components specified with the CONTROL language NEW DATA BASE IS command. The maximum number of components that can be specified is 10,000, and the default is 430.

name is a component name using 1 to 250 characters. Each component name must be unique within a database. It cannot be a reserved word or character. For a list of reserved and restricted words and characters, see Appendix C. In addition, the current system separator cannot appear in the component name. The default system separator is the asterisk (*), which you can change with the system-wide SEPARATOR IS command.

When you are initially assigning component names, consider choosing names that are programming-language specific to avoid changes later.

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). To disable uppercase translation, set the execution parameter OPT043 to YES.

DATE
DECIMAL
DOUBLE
INTEGER
MONEY
REAL specifies the function type. The default is DECIMAL.

delimiter is a single special character to be used as the expression delimiter. The delimiter must appear at both ends of the function expression and cannot appear within the expression. It cannot be the current system separator or command terminator. For a list of special characters, see Appendix B.

Synonyms and Abbreviations:

DECIMAL = DEC
DOUBLE = DOUBLE PRECISION
INTEGER = INT
REAL = FLOAT

expression defines an arithmetic expression containing one or more schema item names or component numbers, constants, system functions (AVG, COUNT, MAX, MIN, SUM, SIGMA), arithmetic operators (+, -, *, /), other functions or strings, the system string *FTODAY*, parameters, parenthetical arithmetic expressions, or Collect File items called by retrieval commands. Note that each system function within a stored function must be enclosed by parentheses.

Considerations

- SYSTEM 2000 software uses a function's name as a default component label for QUEST retrieval output.

- The command stream syntax is not examined by the DEFINE processor. All characters are retained for use when the function is executed, up to a maximum of 250 characters and including blanks (even at the end of a line). Characters in the command stream become meaningful only when the function is called later for processing.

- To use a component in functions whose name includes an arithmetic symbol (+, -, /, or *), refer to the component by its number.

- Function definitions occupy space in the Extended Field Table as follows:

  - A new function uses new Extended Field Table space.

  - A modified function releases old space and uses new space if no restructuring is involved due to other commands in the DEFINE session.

  - A deleted function releases space.

- Released space is not used until the next restructure or reload process occurs. Minimal Extended Field Table space is used after definition modification if either of the following occurs:

  - No data are loaded.

  - Restructuring occurs.

Examples

These examples illustrate function definitions.

Example 1

The following command defines a function that calculates years of employment by subtracting the item START DATE from the item END DATE:

```
3004*YRS EMPLOYED (INTEGER FUNCTION $ ((END DATE-START DATE)/365.25)$):
```

Example 2

The following command defines a function that calculates deductions:

```
3050*DEDUCTION-ADJ (MONEY FUNCTION/(C114 - 5.50)/):
```

Example 3

The following command defines a parametric function where *1* is a parameter to be supplied by the user when the function is invoked.

```
2222*COMMISSION (MONEY FUNCTION & (PAYRATE + *1*)&):
```

Example 4

The following command defines a function that contains a system function:

```
3060*PLUS30 (DATE FUNCTION $((*FTODAY*) + 30)$):
```

Chapter 12

Mapping a Definition: MAP

The MAP command ends a DEFINE session, executes the DEFINE commands, and places the results in the database definition. To execute DEFINE commands, you must issue the MAP command.

After you issue MAP, the software switches to the QUEST processor. A successful mapping increments the definition number by one.

If the MAP command causes restructuring of the database, the data are restructured according to the new definition, and the update cycle is set to one. Restructuring suspends the Update Log.

If issuing a MAP command will cause restructuring of the database, you can also include

- an optional where-clause. A where-clause specifies qualification criteria for selecting data records to be loaded back into the database when restructuring takes place. Only the selected logical entries are restructured; all other logical entries are ignored and eliminated from the database. If the MAP command includes a where-clause that qualifies no data, the update cycle is set to zero.

- an optional ordering-clause if you specify a where-clause. An ordering-clause specifies the order of logical entries when they are loaded back into the database.

If you issue a MAP command containing a where-clause when restructuring is not required, a message appears. The definition remains unchanged, the DEFINE commands are discarded, and the QUEST processor is made available.

Format

```
MAP [[ordering-clause] where-clause] :
```

ordering-clause specifies the order of selected logical entries. You can use only level 0 items. If you specify an ordering-clause, you must also specify a where-clause. For details, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

where-clause selects specific logical entries. You cannot specify the SAME operator. A where-clause is required if an ordering-clause is specified. If you do not specify a where-clause, the entire database is used if restructuring occurs. For details, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

Considerations

- If the RECORD format option is in effect and you issue a MAP command that causes restructuring, only level 0 records are retained. See *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition* for information on the RECORD format option.
- Any limit(s) in effect from the QUEST processor are honored by a MAP command containing a where-clause. For details about the LIMIT command, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

Examples

These examples illustrate the MAP command.

Example 1

The following MAP command causes the execution of preceding commands in the DEFINE session, unless the DISABLE EXECUTION or STOP AFTER SCAN IF ERRORS OCCUR command is in force:

```
MAP:
```

Example 2

When applied to the EMPLOYEE database, the first MAP command with a where-clause would map logical entries for employees whose hire date is after January 15, 1988. The second MAP command would order the logical entries alphabetically by last name.

```
MAP WHERE HIRE DATE GT 01/15/88:

MAP ORDERED BY LAST NAME WHERE HIRE DATE GT 01/15/88:
```

Example 3

The following example, using an excerpt from the EMPLOYEE database, shows the results of issuing MAP commands that have an ordering-clause and a where-clause. The first MAP command is rejected because the definition changes did not cause restructuring. The second MAP command is successful.

```
DESCRIBE:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             4
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))
```

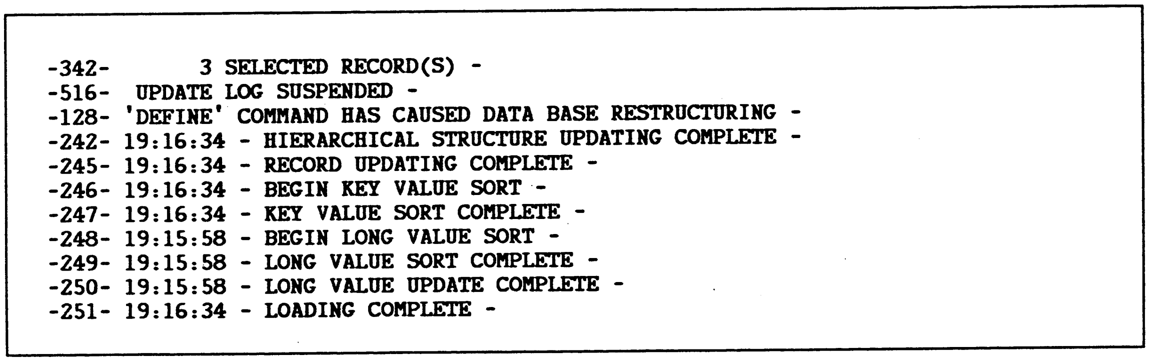
```
PRINT ENTRY:
```

```
1*  1043
2* GIBSON
3* MOLLY I.

1*  1120
2* REID
3* DAVID G.

1*  1265
2* SLYE
3* LEONARD R.
      .
      .
      .
```

```
DEFINE:
50* JOB SKILLS (RECORD):
51* SKILL TYPE (CHAR X(12) IN 50):
MAP WH C1 GT 2000:
.....C
```

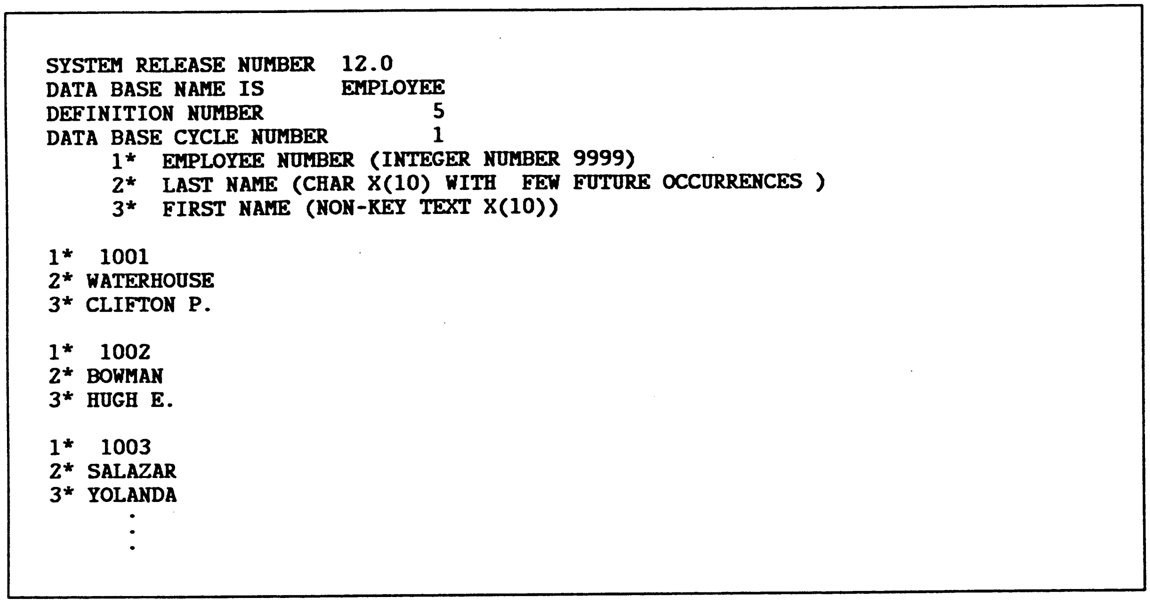
```
 -109- WHERE-CLAUSE ALLOWED ONLY WHEN CHANGES CAUSE RESTRUCTURING -
```

```
DEFINE:
CHANGE 3 TO 3*FIRST NAME (NON-KEY TEXT X(10)):
MAP ORDERED BY C1 WH C1 LT 5000:
```



```
-342-      3 SELECTED RECORD(S) -
-516-  UPDATE LOG SUSPENDED -
-128- 'DEFINE' COMMAND HAS CAUSED DATA BASE RESTRUCTURING -
-242- 19:16:34 - HIERARCHICAL STRUCTURE UPDATING COMPLETE -
-245- 19:16:34 - RECORD UPDATING COMPLETE -
-246- 19:16:34 - BEGIN KEY VALUE SORT -
-247- 19:16:34 - KEY VALUE SORT COMPLETE -
-248- 19:15:58 - BEGIN LONG VALUE SORT -
-249- 19:15:58 - LONG VALUE SORT COMPLETE -
-250- 19:15:58 - LONG VALUE UPDATE COMPLETE -
-251- 19:16:34 - LOADING COMPLETE -
```

```
DESCRIBE:
PRINT ENTRY:
```



```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            5
DATA BASE CYCLE NUMBER       1
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
     3*  FIRST NAME (NON-KEY TEXT X(10))

1*  1001
2* WATERHOUSE
3* CLIFTON P.

1*  1002
2* BOWMAN
3* HUGH E.

1*  1003
2* SALAZAR
3* YOLANDA
      .
      .
      .
```

Chapter 13

Renumbering a Definition: RENUMBER

The RENUMBER command renumbers all components in a database definition, including strings and functions, consecutively in the order listed in DESCRIBE command output. However, C0 cannot be changed. Since component numbers are stored only in the Definition Table, changing them is a simple operation.

When you issue the RENUMBER command, the first component below C0 in DESCRIBE output is assigned the number you specify in the command. Each subsequent component number is then incremented by the number you specify. Schema records and schema items are renumbered first, then functions, then strings.

No component number can be greater than 9999. For example, the following RENUMBER command would not be accepted for a definition having more than four components:

```
RENUMBER STARTING WITH 500 INCREMENTING BY 2000:
```

Format

```
RENUMBER [STARTING WITH number [INCREMENTING BY n]] :
```

number is a positive number from 1 to 9999 to be assigned to the first component in DESCRIBE output after C0. The default is 1.

When a schema record number is changed, all schema record specifications (IN *sr-number*) for member items and descendant records are changed automatically by the software.

n specifies the increment for renumbering. The number can be a number from 1 to 9998. You cannot use 0, and the default is 1.

Consideration

The RENUMBER command does not change component numbers within stored strings and functions. Use the CHANGE command for such changes, or delete the string and function and define a new one.

Examples

These examples illustrate the RENUMBER command, using an excerpt from the EMPLOYEE database. Notice that the component numbers within strings (or functions) are not renumbered.

Example 1

The following example uses all RENUMBER command defaults:

```
DESCRIBE: DESCRIBE STRINGS: DESCRIBE FUNCTIONS:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            8
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)
   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 )
     105*  START DATE (DATE IN 100)
     106*  END DATE (NON-KEY DATE IN 100)
1100*  EMP-NO (STRING ( PRINT C1: ))
1000*  LIST OF DATES (STRING ( LIST C1, C4, OB C4 WHERE HIRE DATE GT 01
       /01/91: ))
  3004*  YRS EMPLOYED (INTEGER FUNCTION ( ((END DATE-START DATE) /365.2
         5)))
```

```
DEFINE: RENUMBER: MAP:
DESCRIBE: DESCRIBE STRINGS: DESCRIBE FUNCTIONS:
```

```
SYSTEM RELEASE NUMBER  11.6A
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            9
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*  POSITION WITHIN COMPANY (RECORD)
       6*  POSITION TITLE (NON-KEY CHAR X(10) IN 5)
       7*  DEPARTMENT (CHAR X(14) IN 5 WITH SOME FUTURE OCCURRENCES )
       8*  START DATE (DATE IN 5)
       9*  END DATE (NON-KEY DATE IN 5)
   11*  EMP-NO (STRING ( PRINT C1: ))
   12*  LIST OF DATES (STRING ( LIST C1, C4, OB C4 WHERE HIRE DATE GT 01
        /01/91: ))
     10*  YRS EMPLOYED (INTEGER FUNCTION ( ((END DATE-START DATE) /365.2
          5)))
```

Example 2

The following example specifies values for the RENUMBER command options:

```
DESCRIBE: DESCRIBE STRINGS: DESCRIBE FUNCTIONS:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             8
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)
   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 )
     105*  START DATE (DATE IN 100)
     106*  END DATE (NON-KEY DATE IN 100)
1100*  EMP-NO (STRING ( PRINT C1: ))
1000*  LIST OF DATES (STRING ( LIST C1, C4, OB C4 WHERE HIRE DATE GT 01
       /01/91: ))
  3004*  YRS EMPLOYED (INTEGER FUNCTION ((*FTODAY - C4)/365))
```

```
DEFINE:
RENUMBER STARTING WITH 100 INCREMENTING BY 10:
MAP:
DESCRIBE: DESCRIBE STRINGS: DESCRIBE FUNCTIONS:
```

```
SYSTEM RELEASE NUMBER  12.0
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            10
DATA BASE CYCLE NUMBER        1
   100*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
   110*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
   120*  FORENAME (NON-KEY CHAR X(20))
   130*  HIRE DATE (DATE)
   140*  POSITION WITHIN COMPANY (RECORD)
     150*  POSITION TITLE (NON-KEY CHAR X(10) IN 140)
     160*  DEPARTMENT (CHAR X(14) IN 140 WITH SOME FUTURE OCCURRENCES )
     170*  START DATE (DATE IN 140)
     180*  END DATE (NON-KEY DATE IN 140)
 200*  EMP-NO (STRING ( PRINT C1: ))
 210*  LIST OF DATES (STRING ( LIST C1, C4, OB C4 WHERE HIRE DATE GT 01
       /01/91: ))
  3004*  YRS EMPLOYED (INTEGER FUNCTION ((*FTODAY - C4)/365))
```

Chapter 14

Schema Item Definition

A schema item represents a group of values by naming and defining the group's characteristics. A schema item is the only type of SYSTEM 2000 component that stores data.

A schema item must belong to one and only one schema record. You must define that schema record before you define any descendant records or any of the schema items to be included in the record. For example, schema record C100 must be defined before you can define items C101 and C102, which specify membership in record C100. For details on defining schema records, see Chapter 15, "Schema Record Definition."

To define a schema item, you specify

- a unique component number and name, which labels the item
- the key or non-key status, which determine whether values for the item are indexed
- the item type and picture, which determine how the data are used and displayed
- the schema record to which the item belongs, which determines relationships among the data
- padding, which determines whether the index of values has optional padding.

You do not need to define schema items consecutively; they can appear anywhere in a DEFINE sequence, as long as the parent schema record has been defined. However, the top-to-bottom order of items within a record is determined by the order in which you define them. This order is retained by SYSTEM 2000 software and is used in storing the values in the Data Table. It is also reflected in DESCRIBE command output.

Once the database is populated, the values for the items can be retrieved in this order by simply requesting a display of the schema record(s), or you can explicitly request the display of any one item or multiple items in any order. The order of items is a convenience only; it may be helpful in some applications, for example, when a PLEX program needs a particular order of values in a record. However, even in this case, PLEX allows you to define items in any order within one or more subschema records.

Format

```
number * name ([NON-KEY] |CHARACTER [NUMBER] [picture] [IN sr-number]
                         |DATE
                         |DECIMAL
                         |DOUBLE
                         |INTEGER
                         |MONEY
                         |REAL
                         |TEXT
                         |UNDEFINED

   [WITH |NO    FUTURE OCCURRENCES] ) :
         |FEW
         |SOME
         |MANY
```

number is an integer from 1 through 9999 to be the component number of the schema item. Each component number must be unique within a database. Note that schema items are included in the maximum number of components specified with the CONTROL language NEW DATA BASE IS command. The maximum number of components that can be specified is 10,000, and the default is 430.

name is a name using 1 to 250 characters to be the component name of the schema item. Each component name must be unique within a database. The name cannot be a reserved word or contain a reserved character. For a list of reserved and restricted words and characters, see Appendix C. Also, the component name cannot contain the current system separator. The default system separator is the asterisk (*), which you can change with the system-wide SEPARATOR IS command.

When you are initially assigning component names, consider choosing names that are programming-language specific to avoid changes later.

Abbreviations and Synonyms:

CHARACTER = CHAR
DECIMAL = DEC
DOUBLE = DOUBLE PRECISION
INTEGER = INT
NON-KEY = NK
NUMBER = NUMB, NUM
REAL = FLOAT
UNDEFINED = UNDEF

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). To disable uppercase translation, set the execution parameter OPT043 to YES.

`NON-KEY`

specifies non-key processing. If you do not specify the keyword NON-KEY, the item is a key item by default. Any number of items can be key items.

The key or non-key specification indicates whether an index is to be maintained for an item. If an item is a key item, the software creates and maintains an index of values for the item. The software uses the index to quickly access logical records that are associated with a particular value or values. Non-key items have no indexes, but the values can still be used for qualification criteria. How an index works is explained in **The Index** on page 11.

`CHARACTER`
`DATE`
`DECIMAL`
`DOUBLE`
`INTEGER`
`MONEY`
`REAL`
`TEXT`
`UNDEFINED`

specifies an item type, which identifies how the values for that item are to be stored and displayed. You must specify one; there is no default. SYSTEM 2000 software offers a variety of character and numeric item types and a date item type. For details on each item type, see **Item Type Specification** on page 14-4.

`NUMBER`

for clarity, you can specify the keyword NUMBER after numeric item types: INTEGER, DECIMAL, MONEY, REAL, and DOUBLE.

picture

designates the size and format of the values to be entered; it also determines how values are stored and displayed. Each item type has its own default picture. For details, see **Picture Specification** on page 14-8.

`IN` *sr-number*

specifies the schema record to which the item belongs, with *sr-number* being the component number of a previously defined schema record. Do not include the C (that is, specify 101, not C101). The default is 0, that is, IN 0, which means that the schema item automatically belongs to the C0 schema record (the ENTRY record) at level 0. For details, see **Record Membership Specification** on page 14-11.

`NO`
`FEW`
`SOME`
`MANY`

specifies future occurrences for padding. The default is NO. You can specify padding for key items only. For details, see **Padding Specification** on page 14-11.

Consideration

If you use an arithmetic symbol (+, -, /, or *) in a schema item name and later use the item in a function, you will need to refer to the item by its number.

Item Type Specification

You must specify an item type when you define a schema item. The item type designates how the values for that item are to be stored and displayed, and the manner of storage determines how the values can be used.

SYSTEM 2000 software offers a variety of character and numeric item types and a date item type. Each item type is discussed below.

Character item types Character item types store data consisting of letters, numerals, special characters, and blanks, up to 250 characters. There are three character item types:

CHARACTER stores alphanumeric values with trailing, leading, and extra internal blanks removed. Only a single embedded blank is retained between syntactic units. For example, an input value of ƀƀJOHNƀƀƀSMITHbƀƀƀ is stored as JOHNƀSMITH.

TEXT also stores alphanumeric values, but blanks are not removed. All blanks must appear in qualification criteria when accessing the data. Trailing blanks could cause problems since they do not appear when listed. For example, ƀƀJOHNƀƀƀSMITHƀƀƀ is stored and displayed as ƀƀJOHNƀƀƀSMITHƀƀƀ.

UNDEFINED stores binary bit-string data. UNDEFINED items can contain any of the 256 EBCDIC characters.

CHARACTER and TEXT item types

You cannot use numeric values for character item types in computations. For example, in the EMPLOYEE database, the item ZIP CODE is a CHARACTER item type. However, no calculations can be done with the ZIP codes, even though they consist exclusively of numbers. If calculations are needed, the item has to be designated as a numeric or date item type.

A value stored in a CHARACTER or TEXT item type can exceed its picture (up to 250 characters) if the specified picture is at least X(4). That is, CHARACTER and TEXT item types have overflow capabilities. See **Picture Specification** on page 14-8 for more information on specifying a picture for character item types.

CHARACTER and TEXT items differ only with regard to storing blanks. For CHARACTER values, leading, trailing, and extraneous (more than one embedded) blanks are discarded. For example, if the item LAST NAME is a CHARACTER item, then when the value ƀƀJOHNƀƀƀƀSMITHƀƀƀ is entered, it is stored as JOHNƀSMITH. Values for TEXT items retain all blanks when stored. In other words, blanks are considered significant characters that occupy storage space, and they must be taken into account when you are searching for or retrieving data. For example, assume that JOHNƀƀƀSMITH is a value for the TEXT item LAST NAME. The following commands produce no results because both values searched for contain less than three embedded blanks:

```
PRINT EMPLOYEE NUMBER WHERE LAST NAME EQ JOHNƀSMITH:

PRINT EMPLOYEE NUMBER WHERE LAST NAME EQ JOHNƀƀSMITH:
```

Only the following command prints the EMPLOYEE NUMBER for John Smith:

```
PRINT EMPLOYEE NUMBER WHERE LAST NAME EQ JOHNƀƀƀSMITH:
```

On the other hand, if LAST NAME were a CHARACTER item, all of the above commands would print the desired information.

A TEXT item type also affects output. Suppose that ten spaces are allotted for displaying LAST NAME data in a LIST command. If LAST NAME is a TEXT item for which the values MARYƀƀbSMITH and JOHNƀSMITH have been entered, the output appears as follows:

```
LAST NAME
MARY   SMI
TH
JOHN SMITH
```

You should specify an item as TEXT when storing reports or graphics where control of blank spaces is imperative to maintain the original appearance of the material. For example, assume that GRAPH is a TEXT item for which the following values have been entered:

```
ƀƀƀƀSSSSSSƀƀƀƀƀƀƀ222222ƀƀƀƀƀƀKKKƀƀƀKKƀƀƀ
ƀƀƀSSSƀƀSSSƀƀƀƀƀ222ƀƀ222ƀƀƀƀƀKKKƀƀKKƀƀƀƀ
ƀƀƀSSSƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀ222ƀƀƀƀKKKƀKKƀƀƀƀƀ
ƀƀƀƀSSSƀƀƀƀƀƀƀƀƀƀƀƀƀƀ222ƀƀƀƀƀKKKKKƀƀƀƀƀƀ
ƀƀƀƀƀƀSSSSƀƀƀƀƀƀƀƀƀ222ƀƀƀƀƀƀƀKKKKKƀƀƀƀƀƀ
ƀƀƀƀƀƀƀƀƀSSSƀƀƀƀƀ222ƀƀƀƀƀƀƀƀƀKKKƀKKƀƀƀƀƀ
ƀƀƀƀƀƀƀƀƀƀSSSƀƀƀ222ƀƀƀƀƀƀƀƀƀƀKKKƀƀKKƀƀƀƀ
ƀƀƀSSSƀƀƀSSSƀƀƀƀ222ƀƀƀƀƀƀƀƀƀƀKKKƀƀƀKKƀƀƀ
ƀƀƀƀSSSSSSƀƀƀƀƀƀ222222222ƀƀƀƀKKKƀƀƀƀKKƀƀ
```

When the values are displayed in the same order as they were entered, the output appears as follows.

```
    SSSSSS       222222      KKK   KK
   SSS  SSS     222  222     KKK  KK
   SSS                222    KKK KK
    SSS              222     KKKKK
      SSSS         222       KKKKK
         SSS     222         KKK KK
          SSS   222          KKK  KK
   SSS   SSS    222          KKK   KK
    SSSSSS      222222222    KKK    KK
```

UNDEFINED item type

For UNDEFINED values, all 256 EBCDIC characters from X'00' through X'FF'are valid. (CHARACTER and TEXT values are restricted to the set of hexadecimal characters greater than X'10'.) 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).

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 shorter than the item's picture. That is, overflow is not allowed. See **Picture Specification** on page 14-8 for more information on specifying a picture for an UNDEFINED item type.

If the 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 is referred to 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' is 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. Note 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 input, SYSTEM 2000 software stores every space, including trailing spaces, as hexadecimal 40.

For either input format, the system separator and entry terminator word must still be given when required, for example,

```
IT CO * 0 EQ 1*X'E2E8E2E3C5D440F2F0F0F0' * 2* value * END* ...

CHANGE C1 EQ X'E2E8E2E3C5D440F2F0F0F0' * WHERE ...
```

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 (*), results are unpredictable. Also, HEX notation could cause errors if the system separator is set to be the single quote.

Date item type You can specify calendar dates using the DATE item type. The default picture for dates is 9(7).

The DATE item type stores calendar dates in a fixed format. The format MM/DD/YYYY is the default format, which means that the value stored must be in the form 07/04/1989 (including the slashes). You can change the format with the QUEST language DATE FORMAT IS command. See *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

If only two numbers are used for the year, SYSTEM 2000 software assumes 19YY until the century changes. For computations involving dates, the value is considered to be the number of days elapsed since 10/15/1582 (day zero).

You can store dates between October 15, 1582 (the advent of the Gregorian calendar) and December 31, 2499. If you need to store dates before 10/15/1582, you can store them as CHARACTER item type values, but they cannot be used in calculations.

If you are interested in information on the Gregorian calendar, see **The Gregorian Calendar and SYSTEM 2000 Dates** on page E-8.

Numeric item types If all values for an item consist of digits only, SYSTEM 2000 software can store those values in a manner suitable for computation. There are five numeric item types: INTEGER, DECIMAL, MONEY, REAL (or FLOAT), and DOUBLE.

Three of the numeric item types have a picture associated with them: INTEGER, DECIMAL, and MONEY. The picture specifies the number of places required by the longest expected value for the item. Numeric values cannot exceed their specified picture; that is, overflow is not allowed for numeric values. See **Picture Specification** on page 14-8 for more information on specifying a picture for numeric item types.

Since values for numeric item types are stored in packed decimal format, a plus (+) or minus (-) sign is automatically associated with each value; therefore, all characters in the picture can be used as digits. You do not need to specify a position for a plus or minus sign.

The numeric item types are discussed individually below.

INTEGER

stores whole numbers (that is, any series of numerals from 0 - 9) and an optional plus or minus sign. For example, in a library database, the item NUMBER OF BOOKS BORROWED would be an INTEGER item type since the values for that item can only be whole numbers. The following chart shows an example of how values for an INTEGER item are entered, stored, and displayed. The item's name is TEMPERATURE, and it has a picture of 99.

Value Entered	Value Stored	Value Displayed
89	+89	89
-17	-17	-17
102 *(Rejected: exceeds picture.)*		

DECIMAL

stores numbers with a decimal point and an optional plus or minus sign. If a decimal point is not explicitly given in an input DECIMAL value, the picture for the item governs the placement of the decimal point. DECIMAL values can have up to 10 digits on either side of the decimal point, but no more than a total of 15 digits. The values are stored in packed decimal format.

For example, the item GRADE POINT AVERAGE takes decimal values. The following chart shows how values for that item appear when entered, stored, and displayed. The item GRADE POINT AVERAGE has a picture of 9.999.

Value Entered	Value Stored	Value Displayed
2.345	+2345	2.345
2.3	+2300	2.300
2.3456	+2345 *(truncated)*	2.345
2345 *(implied decimal)*	+2345	2.345
23 *(implied decimal)*	+23	0.023
12.234	*(rejected: exceeds picture)*	
-2.345	-2345	-2.345

MONEY

also stores numbers with a decimal point. These values include a floating dollar sign at the left, CR (for credit, if negative) at the right, and a comma (if appropriate), regardless of the picture or how the value was entered. Like DECIMAL values, MONEY values can have up to ten digits on either side of the decimal point, but no more than a total of fifteen digits. The values are stored in packed decimal format.

For example, consider the following examples for item ACCOUNT BALANCE, which has a picture of $9999.99.

Value Entered	Value Stored	Value Displayed
1234.56 CR	-123456	$1,234.56 CR
$12.46	1246	$12.46
12345.78	*(rejected: exceeds picture)*	
$-123.45	-12345	$123.45 CR
9 *(implied decimal)*	9	0.09

REAL (or FLOAT) stores fullword (single precision) floating point numbers. REAL items do not have a picture. Each REAL value occupies one word (four bytes) in the database.

DOUBLE stores double word (double precision) floating point numbers. DOUBLE items do not have a picture. Each DOUBLE value occupies two words (eight bytes). You can also use the DOUBLE item type for storing time. (SYSTEM 2000 software does not have a TIME item type.)

REAL and DOUBLE items contain floating point values. 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.

The output format for REAL and DOUBLE values is E-notation, that is, ±*n.n*E±*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.

Picture Specification

The picture for an item designates the size and format of the values for the item and determines how values are stored and displayed. Each item has its own default picture, or you can specify one.

Illus. 14.1 on page 14-10 provides picture specification information for each item type discussed below.

Pictures for character item types The picture for character item types (CHARACTER, TEXT, and UNDEFINED) indicates the number of places it takes to store a value for that item. For example, storing values for the item MONTH usually requires nine places, storing values for the item LAST NAME usually requires ten places, and so on.

Pictures for character item types can be from one to 250 characters long, specified with the character X (up to XXXXX) or using the format X(*n*) where *n* is 1 to 250. The default picture for CHARACTER and TEXT items is X(7). The default picture for UNDEFINED items is X(4).

The picture for CHARACTER and TEXT items should be the number of places that would accommodate most of the values for an item. That is, values stored in CHARACTER and TEXT item types can exceed their picture (up to 250 characters) if the specified picture is at least X(4). That is, CHARACTER and TEXT item types have overflow capabilities. The

UNDEFINED item type, however, does not allow overflow; therefore, the length of each UNDEFINED value must be equal to or less than its picture.

When a stored value for CHARACTER and TEXT items is longer than the picture, the value is divided into two parts. The first part is kept in the same table as the other shorter values. The second part is stored in another table, called the Extended Field Table.

Assume that values SMITH and FAULKENBERRY are to be stored for LAST NAME, which has a picture of X(10). The following example shows how the values appear when entered, stored, and displayed:

Value Entered	Value Stored in Data Table	Value Stored in Extended Field Table	Value Displayed
SMITH	S M I T H _ _ _ _ _		SMITH
FAULKENBERRY	F A U L K E _ _ _ _	N B E R R Y	FAULKENBERRY

Pictures for dates The default picture for the DATE item type is 9(7), and you cannot specify any other picture.

Pictures for numeric item types The picture for numeric item types INTEGER, DECIMAL, and MONEY specifies the number of places required to store a value for that item. REAL and DOUBLE item types do not have picture specifications.

For example, a value for the item YEARS OF EXPERIENCE requires at most two places, a value for the item WEIGHT OF EMPLOYEE requires three places, and so on. The picture must allow for the longest expected value for the item, as overflow is not allowed for numeric values.

The picture specifications can be from one to fifteen digits long with an optional plus or minus sign, specified with the numeral 9 (up to 99999) or using the format 9(*n*) where *n* is 1 to 15. The default picture for an INTEGER item is 9(7), and the default for DECIMAL and MONEY items is 9(6).9(2).

For DECIMAL and MONEY items, if the picture is of the format 9(n_1).9(n_2), then n_1 and n_2 must each be greater than or equal to zero and less than or equal to ten. Their sum must be equal to or less than fifteen. That is, the sum of n_1 and n_2 must be less than sixteen.

Values for INTEGER items that are longer than the picture are rejected. For DECIMAL and MONEY items, values with more places to the left of the decimal point than specified in the picture are rejected, while values with more places to the right of the decimal point are truncated.

Illus. 14.1 Picture Specifications for Item Types

Item Type	Acceptable Pictures	Default Picture	Maximum Picture	Examples

CHARACTER and TEXT	X
	XX
	XXX
	XXXX
	XXXXX
	X(n) where $1 \leq n \leq 250$
		X(7)	X(250)	X
				X(3)
				X(25)
				XXX

UNDEFINED	X
	XX
	XXX
	XXXX
	XXXXX
	X(n) where $1 \leq n \leq 250$
		X(4)	X(250)	X
				X(3)
				X(25)
				XXX

DATE	none	9(7)		

INTEGER	9
	99
	999
	9999
	99999
	9(n) where $1 \leq n \leq 15$
		9(7)	9(15)	9
				9(3)
				9(10)
				999

DECIMAL	9.9 * 99.99 999.999 9999.9999 99999.99999 9(n_1).9(n_2) where $0 \leq n \leq 10$	9(6).9(2)	$n_1 + n_2 \leq 15$	.9 9.9 99.9(4) 9(3).99 9(10).9(3)
MONEY	same as DECIMAL but includes an optional $	9(6).9(2)	$n_1 + n_2 \leq 15$	.9 9.9 $99.9(4) 9(3).99 $9(10).9(3)
REAL	none	single word		
DOUBLE	none	double word		

* For DECIMAL and MONEY pictures, you can enter any combination of 9s, for example, 9.99, 9(4).9, 99.9999, and so on.

Record Membership Specification

When defining a schema item, you must identify the schema record to which it belongs. This is referred to as the *IN specification*. Otherwise, SYSTEM 2000 software assumes that the item belongs to the ENTRY (level 0) record.

You specify the schema record to which an item belongs by using the keyword IN followed by the component number of the schema record. Also, the keyword IN designates that a schema record is a child of another schema record.

You can specify record membership before or after specifying padding. For example, either of the following forms is acceptable:

```
101* POSITION TITLE (CHAR WITH MANY FUTURE OCCURRENCES IN 100):

101* POSITION TITLE (CHAR IN 100 WITH MANY FUTURE OCCURRENCES):
```

A schema record can have any number of member items, as long as the total number of components in the database is within the maximum specified when the database was created.

You do not need to define items consecutively below their appropriate schema record; the IN specification determines the item's record membership. However, the items remain in the order defined under the schema record.

Padding Specification

The padding specification specifies future occurrences for padding of key items. Padding reserves contiguous space for later entries in the Multiple Occurrence Table when a new index page is created.

This multiple occurrence padding also affects distinct values padding if padding for the Distinct Values Table is enabled for the database with the QUEST language command ENABLE VALUES PADDING. Distinct Values Table padding is calculated inversely from the multiple occurrence padding. If Distinct Values Table padding is enabled, consider the multiple occurrence padding carefully for each item. For information on the ENABLE VALUES PADDING command, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

The index space reserved for multiple occurrences of a value for an item is indicated below.

- NO - 0% padding
- FEW - 100% padding
- SOME - 400% padding
- MANY - 1400% padding.

Use of padding can reduce I/O required to process qualification criteria in update or retrieval commands.

Examples

These examples illustrate defining schema items.

Example 1

The following schema item definitions use defaults:

```
1* EMPLOYEE NUMBER (INTEGER):

2* LAST NAME (CHAR):

5* BIRTHDAY (DATE):
```

Because record membership is not specified, these items are in the C0 schema record (the ENTRY record). They are all key items with no padding, and they have default pictures as shown in the following DESCRIBE output:

```
DESCRIBE:
```

```
1* EMPLOYEE NUMBER (INTEGER NUMBER 9(7))
2* LAST NAME (CHARACTER X(7))
5* BIRTHDAY (DATE)
```

Example 2

The following schema item definitions specify one or more options:

```
1* EMPLOYEE NUMBER (INTEGER 9(5)):

2* LAST NAME (CHAR X(20) WITH SOME FUTURE OCCURRENCES):

100* POSITION WITHIN COMPANY (RECORD):

104* DEPARTMENT (CHAR X(14) IN 100):

101* POSITION TITLE (CHAR X(7) IN 100):
```

Example 3

The following two definitions are equivalent because the default picture for INTEGER item types is 9(7):

```
1* EMPLOYEE NUMBER (INTEGER):

1* EMPLOYEE NUMBER (INTEGER 9(7)):
```

Example 4

The following definitions, which specify DECIMAL item types and pictures, are equivalent:

```
11* ACCRUED VACATION (DECIMAL 999.99):

11* ACCRUED VACATION (DECIMAL 9(3).9(2)):
```

Example 5

The following definition defines a date, which never has a picture specification:

```
4* HIRE DATE (DATE):
```

Example 6

The following definitions define MONEY, CHARACTER, and TEXT item types. In each case, the schema record must have been defined before the item definition.

```
124* GROSS PAY (NON-KEY MONEY $9999.99 IN 120):

411* SCHOOL (CHAR X(15) IN 400):

303* COMMENT (NON-KEY TEXT X(5) IN 300):
```

Example 7

The following definitions use the default record membership for items, which is the level 0 record:

```
1* EMPLOYEE NUMBER (INTEGER NUMBER 9999):

2* LAST NAME (CHAR X(10)):
```

Example 8

The following definitions specify record membership for the items, where schema record C100 has already been defined:

```
101* POSITION TITLE (NON-KEY CHAR X(10) IN 100):

102* DEPARTMENT (CHAR X(14) IN 100):
```

Example 9

The following definitions specify key items (the default) and no padding:

```
2* LAST NAME (CHAR X(12)):

101* POSITION TITLE (CHAR IN 100):
```

Example 10

The following definitions specify key and non-key items without padding:

```
1* EMPLOYEE NUMBER (KEY INTEGER 9(5)):

102* DEPARTMENT (KEY CHAR X(14) IN 100):

3* FORENAME (NON-KEY CHAR X(20)):
```

Example 11

The following definitions specify key items with padding:

```
7* GENDER (KEY CHAR X(6) WITH MANY FUTURE OCCURRENCES):

8* ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES):
```

Example 12

The following definitions show various combinations of the syntax for defining REAL, DOUBLE, and UNDEFINED items:

```
 1* ITEMA (REAL):
 2* ITEMB (FLOAT):
 3* ITEMC (NON-KEY FLOAT):
 4* ITEMD (KEY DOUBLE WITH FEW FUTURE OCCURRENCES):
 5* ITEME (DOUBLE PRECISION):
 6* ITEMF (UNDEFINED X(8)):
 7* ITEMG (NON-KEY UNDEF):
 8* ITEMH (NK REAL NUMBER):
 9* ITEMI (KEY UNDEF X(100)):
10* RECORD1 (RECORD):
11* ITEMJ (REAL IN 10):
12* ITEMK (UNDEF IN 10 WITH SOME FUTURE OCCURRENCES):
13* ITEML (NK UNDEFINED X(60) IN 10):
14* ITEMM (NON-KEY DOUBLE PRECISION NUM IN 10):
```

Chapter 15

Schema Record Definition

A schema record represents a class of data records. That is, a schema record is a group of schema items that are treated as a unit. To define a schema record, you specify a component number and name (which are labels) and the parent-child relationships among this record and others. These relationships establish the hierarchical structure of a SYSTEM 2000 database.

Except for the C0 (ENTRY) record, which is automatically created, you must define a schema record before defining any descendant records or any of the schema items for the record. For example, you must define schema record C100 before defining its items and before specifying C100 as a parent record for its other lower level records. However, you do not have to define descendant records in any order.

You can define up to 32 levels of schema records (including level 0). The limit on the number of schema records at a single level or on the number of descendant schema records a given schema record can have is the maximum number of components allowed for a database.

You do not have to define schema records in logical order (as long as the parent record has been previously defined). SYSTEM 2000 software places schema records in logical order below the proper parent record. This logical order is reflected in the Definition Table and in DESCRIBE command output. The left-to-right order of schema siblings is determined by the order in which you define schema records. This order is preserved in the Definition Table and is reflected in DESCRIBE output; however, DESCRIBE output does not determine the order of entering data. You specify the input order of data records when the data are loaded, and it is retained by the software in the Hierarchical Table. You can print records and values in logical order or any order requested. For discussions of loading, retrieving, and updating a database, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*.

Format

```
number * name (SCHEMA RECORD [IN sr-number]) :
```

number is an integer from 1 through 9999 to be the component number of the schema record. Each component number must be unique within a

Abbreviations: SCHEMA RECORD = RECORD, SR database. Note that schema records are included in the maximum number of components specified with the CONTROL language NEW DATA BASE IS command. The maximum number of components that can be specified is 10,000, and the default is 430.

name is a component name using 1 to 250 characters. Each component name must be unique within a database. It cannot be a reserved word or character. For a list of reserved and restricted words and characters, see Appendix C. Also, the current system separator cannot appear in the component name. The default system separator is the asterisk (*), which you can change with the system-wide SEPARATOR IS command.

When you are initially assigning component names, consider choosing names that are programming-language specific to avoid later changes.

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). To disable uppercase translation, set the execution parameter OPT043 to YES.

IN *sr-number* is the record component number (previously defined) to which the new record belongs. Do not include the C (for example, specify 100, not C100).

For all schema records other than those at level 1, the IN *sr-number* specification is required. If omitted, the schema record becomes a level 1 record (IN 0), by default.

Consideration

If you use an arithmetic symbol (+, -, /, or *) in a schema record name and later use the record in a function, you will need to refer to the record by its number.

Examples

These examples illustrate defining schema records.

Example 1

The following command defines a level 1 record because there is no IN specification. You can define level 1 schema records at any time, because the level 0 record exists at the time the database is named.

```
100* POSITION WITHIN COMPANY (SCHEMA RECORD):
```

Example 2

The following commands define two level 2 schema records. The records are siblings, because they are both descendants of the C100 record at level 1. In DESCRIBE command output, the C110 record would be displayed before the C130 record. The C100 record must have been defined prior to these level 2 records. The IN specification in both commands is required.

```
110* SALARY WITHIN POSITION (RECORD IN 100):

130* ADDITIONAL INFORMATION (SR IN 100):
```

Chapter 16

Stopping If Errors: STOP AFTER SCAN IF ERRORS OCCUR

The STOP AFTER SCAN IF ERRORS OCCUR command prevents mapping of all DEFINE commands if any error occurs. You can then edit the commands to correct any error before mapping the definitions into the Definition Table. If no errors are found, mapping takes place.

STOP AFTER SCAN IF ERRORS OCCUR allows syntactic scanning of all DEFINE commands. If one or more errors exist in the DEFINE session when you issue the MAP command, a message is displayed, all DEFINE commands are discarded, and the QUEST processor is made available for subsequent commands. Definitions discarded under STOP AFTER SCAN IF ERRORS OCCUR must be corrected, if necessary, and resubmitted before they can be mapped.

You can issue a STOP IF command, which performs the same function, from the QUEST and QUEUE processors.

The default is that this command is not in effect. You can set it at the beginning of a Self-Contained Facility session; the command remains in effect throughout the session.

Format

```
STOP AFTER SCAN IF ERRORS OCCUR ;
```

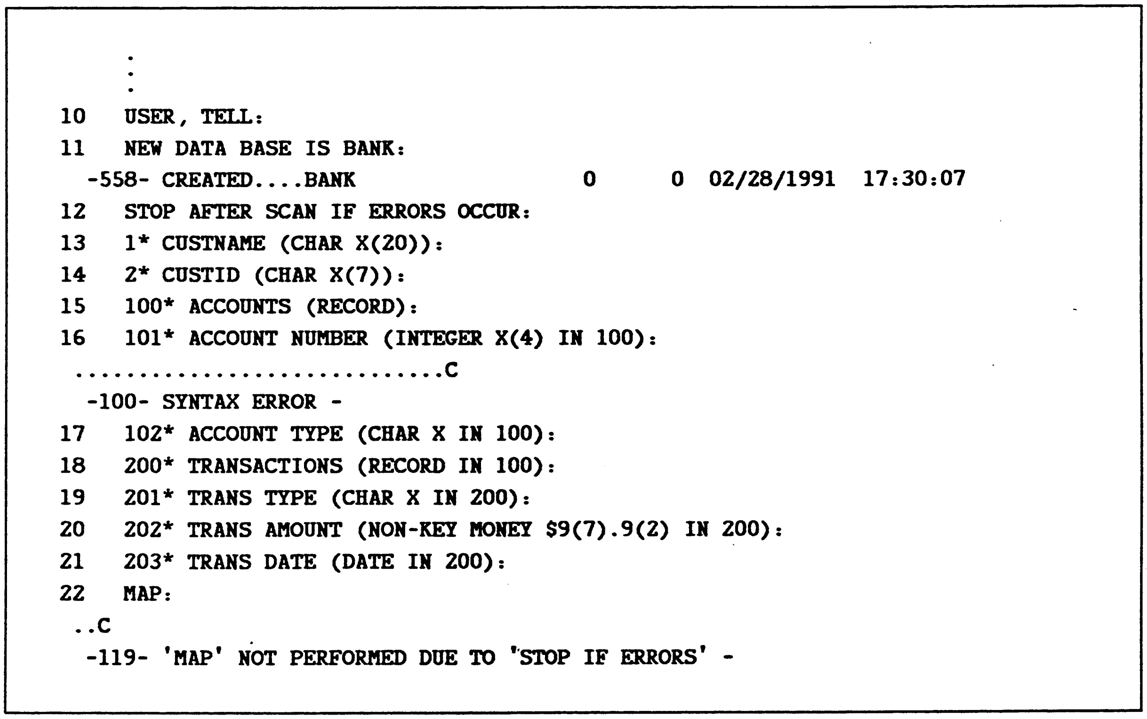
Considerations

- When both STOP AFTER SCAN IF ERRORS OCCUR and DISABLE EXECUTION commands have been issued, DISABLE EXECUTION overrides. No mapping occurs until the commands are resubmitted without errors and mapped under the control of ENABLE EXECUTION.
- You can issue this command from the QUEST and QUEUE processors as well as the DEFINE processor. Commands in all three processors are affected, regardless of the processor from which STOP AFTER SCAN IF ERRORS OCCUR is issued.

Example

The following example illustrates the effects of a STOP AFTER SCAN IF ERRORS OCCUR command for the following DEFINE commands. An arrow indicates the DEFINE command containing an error. (The picture for a numeric item type must be specified with a 9; character item types use X.)

```
  USER, TELL:
  NEW DATA BASE IS BANK:
  STOP AFTER SCAN IF ERRORS OCCUR:
  1* CUSTNAME (CHAR X(20)):
  2* CUSTID (CHAR X(7)):
  100* ACCOUNTS (RECORD):
→ 101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
  102* ACCOUNT TYPE (CHAR X IN 100):
  200* TRANSACTIONS (RECORD IN 100):
  201* TRANS TYPE (CHAR X IN 200):
  202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
  203* TRANS DATE (DATE IN 200):
  MAP:
```



```
     .
     .
     .
10   USER, TELL:
11   NEW DATA BASE IS BANK:
  -558- CREATED....BANK                   0      0  02/28/1991  17:30:07
12   STOP AFTER SCAN IF ERRORS OCCUR:
13   1* CUSTNAME (CHAR X(20)):
14   2* CUSTID (CHAR X(7)):
15   100* ACCOUNTS (RECORD):
16   101* ACCOUNT NUMBER (INTEGER X(4) IN 100):
 ..............................C
  -100- SYNTAX ERROR -
17   102* ACCOUNT TYPE (CHAR X IN 100):
18   200* TRANSACTIONS (RECORD IN 100):
19   201* TRANS TYPE (CHAR X IN 200):
20   202* TRANS AMOUNT (NON-KEY MONEY $9(7).9(2) IN 200):
21   203* TRANS DATE (DATE IN 200):
22   MAP:
 ..C
  -119- 'MAP' NOT PERFORMED DUE TO 'STOP IF ERRORS' -
```

Chapter 17

String Definition

Strings provide a means of storing an entire command, part of a command, or a series of commands for later use. This type of string is referred to as a *stored string.* By invoking a stored string, you request the processing of update commands, retrieval commands, and other commands without repeating the full command syntax.

You invoke a string by issuing its component name or number either by itself or as part of an action-clause or where-clause of a QUEST or QUEUE retrieval or update command. For information on invoking stored strings, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition.*

Strings provide convenience and security. Because strings are created once, then stored in a database definition, your retrievals and updates are not prone to coding errors or errors in command construction.

Strings can contain

- parameters
- calls to other functions or strings
- system strings (*NOW*, *TODAY*, *FTODAY*, *DATA*).

There are two kinds of strings: strings that are used exactly as defined by issuing their component name or number, and *parametric strings* that have parameters for which values must be provided when the string is invoked.

You can specify up to 31 levels of nesting, 31 parameters, any combination of the two yielding a total of 31, or a maximum of 1200 characters of string expansion and/or parameters. Parameters can be supplied within a string definition calling another string or function with parameters.

Strings are subject to the same security measures as other components. A secondary password may have authority to use only a specified set of strings. Also, the password used when a string is invoked must have proper authority for all components referenced in the stored command.

Format

```
number * name (STRING delimiter command-stream delimiter) :
```

number is an integer from 1 through 9999 to be the component number of the string. Each component number must be unique within a database. Note that strings are included in the maximum number of components specified with the CONTROL language NEW DATA BASE IS command. The maximum number of components that can be specified is 10,000, and the default is 430.

name is a component name using 1 to 250 characters. Each component name must be unique within a database. It cannot be a reserved word or character. For a list of reserved and restricted words and characters, see Appendix C. The current system separator cannot appear in a string name. The default system separator is the asterisk (*), which you can change with the system-wide SEPARATOR IS command.

When you are initially assigning component names, consider choosing names that are programming-language specific to avoid later changes.

Note that, by default, the software translates any component name entered in lowercase to all uppercase before processing (except when an alternate Command File or Data File is used to enter the definition). To disable uppercase translation, set the execution parameter OPT043 to YES.

delimiter is a single special character to be used as the command stream delimiter. The delimiter must appear at both ends of the command stream and cannot appear within the command stream. It cannot be the current system separator or command terminator. For a list of special characters, see Appendix B.

command-stream defines a SYSTEM 2000 command stream using 1 to 250 characters. The command stream can consist of partial commands, one command, or several commands. Also, it can contain parameters, calls to other strings or functions, or system strings (*NOW*, *FTODAY*, *TODAY*, *DATA*).

Considerations

- The command stream syntax is not examined by the DEFINE processor. All characters are retained for use when the string is executed, up to a maximum of 250 characters and including blanks (even at the end of a line). Characters in the command stream become meaningful (for example, commands or parts of commands) only when the string is called later for processing.

- If you use an arithmetic symbol (+, -, /, or *) in a string name and later use the string in a function, you will need to refer to the string by its number.

- String definitions occupy space in the Extended Field Table as follows:
    - A new string uses new Extended Field Table space.
    - A modified string releases old space and uses new space if no restructuring is involved due to other commands in the DEFINE session.
    - A deleted string releases space.
- Released space is not used until the next restructure or reload process occurs. Minimal Extended Field Table space is used after definition modification if either of the following occurs:
    - No data are loaded.
    - Restructuring occurs.

Examples

These examples illustrate string definitions.

Example 1

The following command defines a string using a slash (/) as the delimiter:

```
1100* EMP-NO (STRING / PRINT C1: /):
```

Example 2

The following string definition defines a parametric string, where *1* is a date parameter to be supplied by the user when the string is invoked. This definition uses a dollar sign ($) as the delimiter; the default slash cannot be used as a delimiter because it will be used in the supplied date.

```
1000* LIST OF DATES (STRING $ LIST C1, C4, OB C4
   WHERE HIRE DATE GT *1*: $):
```

Appendices

Collating Sequence and Character Set

Special Characters

Reserved and Restricted Words and Characters

Reserved File Names

Supplementary Topics

Appendix A

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 B

Special Characters

These are the special characters that you can use in SYSTEM 2000 commands:

¢

.

<

(

+

|

&

!

$

*

)

;

¬

-

/

%

_

>

?

#

@

'

=

"

Appendix C

Reserved and Restricted Words and Characters

You cannot use the following 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 next two sets of 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 D

Reserved File Names

The following file names are reserved:

S2KSNAP
S2KMSG
S2KCOMP
SYSUDUMP
SYSABEND
SYSOUT
S2KUSERS
TAPES2K
KEEPFILE
STEPLIB
SYSLIB
S2KPARMS
S2KPAD*nn*

S2KSYS00
S2KSYS01
S2KSYS02
S2KSYS03
S2KSYS04
S2KSYS05
S2KSYS06
S2KSYS07
S2KSYS08

S2KSYS10
S2KSYS11
S2KSYS12
.
.
.
and so forth

SF01
SF02
SF03
SF04
SF05
SF06

SF11
SF12
SF13
.
.
.
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.

Appendix E

Supplementary Topics

THE DEFINE PROCESSOR IN A MULTI-USER/MULTI-THREAD ENVIRONMENT

In a Multi-User environment, the DEFINE processor can be called only if the database is open for exclusive use. The processing of the command stream for the DEFINE session ties up one thread until the MAP processing is complete. Therefore, there is no multi-processing within SYSTEM 2000 software if there is only one thread and it is being used for a DEFINE session.

In a Multi-Thread environment, several threads can each hold a DEFINE session concurrently.

TP PROCESSING OF DEFINE SESSIONS

A teleprocessing (TP) segment consists of one or more commands in any Self-Contained Facility language. A TP segment begins with the first character transmitted and ends with a message completion signal for the hardware system, such as carriage return, EOB (end of block), EOT (end of transmission), or C . (One thread is tied up for the duration of a segment.)

Up to 1200 characters can be entered for a single TP segment, including the DEFINE and MAP commands. *

A DEFINE session cannot span TP segments. If a MAP command is not issued within a segment, DEFINE commands in the segment are lost. If subsequent segments contain additional DEFINE commands, they are not processed as expected since the software has no knowledge of commands from prior segments. Depending on what is being done, each command is rejected or seemingly accepted. Overall results when MAP is issued are not correct, even if no error messages are produced.

User Exits

User exits permit execution of code you have supplied to process data from a database during execution of SYSTEM 2000 software. If user exits are used, the STEPLIB specifications in the JCL must specify a library for those user-exit interfaces and routines that are to be dynamically loaded. See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for discussions of user exits in general.

PROCESSING LARGE DEFINITIONS

The following considerations are relevant to processing large definitions:

- Entering component definitions in logical order can save a large amount of processing time. See **Processing Order for Definitions** on page 23.

- After the database is defined and populated, SYSTEM 2000 commands can reference components by name or number. Consider the following processing costs when selecting the final numbering scheme for a definition:

    - If the component numbers are in numeric order, each command that references a component by its component number causes the component number to be converted to an integer number for internal use.

    - If component numbers vary from numeric order, a command that references a component by its number causes the number to be converted to an integer, then conversion is followed by a binary search of a list of all component numbers to produce a number for internal use.

* The software accepts 1200-character segments; however, the TP interface may require smaller segments.

BUFFERS REQUIRED IN A DEFINE SESSION

When you create or modify a database definition, the SYSTEM 2000 buffer manager obtains buffers from the pools that were specified by the POOLn execution parameters. (For details about setting up buffers and pools, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.) The number of buffers required in a DEFINE session depends on three factors:

- MAXCOMP, which determines the maximum number of components that can be defined for the database. You specify MAXCOMP in the NEW DATA BASE IS command. For example, NEW DATA BASE IS COMPANY/5000 specifies a maximum of 5,000 components. The number includes schema items, schema records, strings, and functions. The default is 430. For more details, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

- the LDBSIZE execution parameter, which determines the maximum number of components that can be processed in the current SYSTEM 2000 session. The default is 1000. LDBSIZE determines the largest DEFBLOCK that can be used during this session; it also determines the maximum MAXCOMP for any database that can be opened (or created) in this session. For more information, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

- the buffer size and usage that was specified for each pool. For more information about setting up pools, see the discussion of the POOL*n* execution parameter in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

DEFINE processing consists of two phases. During Phase 1, SYSTEM 2000 software scans the new component definitions (or modifications) and builds a temporary definition. Phase 2 finalizes the definition at MAP time and stores it in database Files 1 and 3.

Phase 1 uses buffers from the pool with the largest buffer size having D, DE, or B usage. The buffer manager uses the following formula to calculate the number of buffers needed:

```
(((MAXCOMP x 28)/bufsiz) + 4 + 2)
```

where *bufsiz* is the number of bytes in the largest buffer having D, DE, or B usage. The additional six buffers do not necessarily come from the same pool. Four of the buffers are for scratch files (S or B usage), and the other two are for database Files 1 and 3 (D, DE, or B usage).

Phase 2 requires the following number of bytes:

```
(LDBSIZE x 4)
```

If one or two buffers (from any pool with any usage) are adequate, the buffer manager uses that pool. If more than two buffers are needed, the buffer manager acquires them from the pool with the largest buffers that were specified for database usage (D, DE, or B).

Note: Buffers acquired during Phase 1 are freed before Phase 2. Therefore, in most situations, the buffers used in Phase 1 can be reused in Phase 2.

For example, if MAXCOMP equals 10,000 and LDBSIZE equals 10,000, the following POOL*n* specification should be sufficient:

```
    POOL0=6216/52/B
```

This pool specification sets up fifty-two buffers having a block size of 6216 with B usage, that is, for either scratch files or database files.

You could also specify these pools:

```
    POOL0=6216/4/S
    POOL1=23464/14/D
```

POOL0 sets up a pool of four buffers having a block size of 6216 for scratch file usage. POOL1 contains fourteen buffers having a block size of 23464 for database usage.

Note: The example and formulas here show absolute minimums. To prevent buffer steals, you should always allocate ten or more buffers beyond the minimum number required for database usage. However, there is no advantage to allocating more scratch file buffers than are required.

In addition, the number of I/Os to the scratch pad is proportional to the block size; the larger block sizes require fewer I/Os. For example, a scratch pad blocked at 6216 bytes experiences about four times the I/Os of a scratch pad blocked at 23464. The number of I/Os affects real time for the job, but it does not affect CPU time.

If the required number of buffers for LDBSIZE is not available when SYSTEM 2000 software is invoked, WTO message 0116 is issued, followed by user abend 101. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for explanations of these error messages.

AUTOMATIC RESTRUCTURING

When a schema change requires restructuring the database, it is done automatically by SYSTEM 2000 software. (No restructuring is required until the database contains values.) This section discusses schema modifications that cause the database to be restructured automatically and how you can optimize the restructuring operations.

Changes That Do Not Cause Restructuring

The following definition changes never cause restructuring:

- changing a component number
- changing a component name
- adding a record at the end of DESCRIBE output order
- changing an item type
- changing padding
- changing a decimal position within a numeric picture
- changing a key item to non-key when the item has no values
- changing a non-key item to key when no record exists for the non-key item
- changing string and function definitions
- adding or deleting strings or functions.

Changes That Cause Restructuring

The following changes to a definition cause an automatic restructure of the database's index and Data Table:

- changing a picture for a character item that exists
- adding or deleting an item to a record that exists
- adding or deleting a record and/or an item in DESCRIBE output order above a record that exists
- changing a key item to non-key when the item has values
- changing a non-key item to key when the item is in a record that exists.

Three types of modification to the schema portion of a database definition can cause restructuring if a data record exists for the schema record or schema item being modified:

- changing the size of a schema record when data records exist (since the last restructure or reload) for that schema record. Adding or deleting an item changes the size of a record. Also, changing the picture of a CHARACTER or TEXT item type when the record exists always causes restructuring.

- causing the position of a record in DESCRIBE output order to change. Schema records (and their items) are stored below their parent records in the Definition Table, and this order is retained and reflected in DESCRIBE output. If a new schema record is added for a parent schema record that is defined (and displayed in DESCRIBE output) in the middle of a schema, restructuring takes place because the rest of the schema rolls down to accommodate the new record. This changes internally set tags for the components below the new record.

  However, a new schema record at level 1 and all of its descendant schema records can be added at any time since it is placed at the bottom of the existing schema. Likewise, a new child for the last schema record in the schema does not cause restructuring.

- Changing the status of an item from key to non-key or vice versa, which causes restructuring in the following situations:

  - changing a key item to non-key when the item contains values

  - changing a non-key item to key when a record (for the item) has existed since the last restructure or reload.

For efficiency and optimization of the restructuring operations, see the next topic.

Optimizing Restructuring Operations

The following suggestions can help optimize the restructuring of your database:

- Restructuring does not take place until commands are mapped. Therefore, if you issue all commands that cause restructuring in one DEFINE session, only one restructuring takes place (when the MAP command is issued), which saves processing time.

- If the database has become so skewed that a RELOAD is advisable, do the needed definition changes with restructuring instead of with the RELOAD command.

- When you need to change an item to key status and other DEFINE commands already require restructuring of the database, it is more efficient to use the DEFINE language CHANGE component command than to use the CONTROL language CREATE/REMOVE INDEX commands. However, if changing the key/non-key status for the item(s) is the only change to be made, use the CREATE/REMOVE INDEX commands; they operate only on the individual item(s) specified. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a detailed discussion of the CREATE/REMOVE INDEX commands.

ALTERNATE METHODS FOR CHANGING A SCHEMA

You can accomplish schema changes (other than those done with the CHANGE and RENUMBER commands) by deleting existing components and defining new ones. However, you should take the following consequences into consideration:

- Restructuring removes stored data for the deleted components.

- Deleting a schema record also removes the corresponding data records, their descendant records, and all values.

Therefore, deleting a record in this manner means that all member items and descendant records must also be redefined if they are to be represented in the new database structure. The new items have no stored values.

If stored values for an existing item are acceptable under a new definition (for example, if the existing item type is numeric and the new item type is character), you can save and re-enter the data.

You can use either PLEX programs or the procedure described below to save and re-enter the data:

1. Before redefining, use the QUEST language UNLOAD command to create a Data File containing the items to be reloaded, along with other data to be used as qualification criteria when the data are re-entered.

2. After redefining, use QUEUE processing, with the appropriate update command, the *DATA* system string, and a where-clause to re-enter the data.

See *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition* for a discussion of the UNLOAD command and the Data File and Report File. See *Technical Report: S2-110 QUEUE Language for SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS* for a discussion of *DATA*. See the *SYSTEM 2000 PLEX Manual, Version 12, First Edition* for a discussion on unloading and loading data with a PLEX program.

THE GREGORIAN CALENDAR AND SYSTEM 2000 DATES

As mentioned in **Date item type** on page 14-6, dates stored in a SYSTEM 2000 database must be equal to or greater than October 15, 1582, the advent of the Gregorian calendar. *But just what is the Gregorian calendar and how does it differ from the Julian calendar?*

In 45 B.C., Julius Caesar created a perpetual cycle of four years, the first three each containing 365 days and the fourth containing 366 days. The practice of adding one day to the month of February every fourth year (Leap Year) is still observed today.

There are approximately 365-1/4 days in a solar year, but not exactly. An exact solar year is 365 days, five hours, 48 minutes, and 47.8 seconds. This difference of approximately eleven minutes became significant over several centuries of time.

In 1582, the final calendar correction was done by Pope Gregory XIII. This calendar, which we use today, is called the Gregorian calendar. Pope Gregory XIII decreed that ten days should be eliminated from 1582 to make up for the extra days that had accumulated since the beginning of the Julian calendar. Thus, in many countries of the world, the day after October 4, 1582, became October 15, 1582.

The other adjustment to the calendar made by Pope Gregory XIII was the installation of the new Gregorian Leap Year rule, which will serve us for more than a thousand years of future time. Ordinarily, a centesimal year (that is, a year ending in 00) would be a leap year. However, the new rule stated that for any centesimal year not exactly divisible by 400, February would contain 28 days. Thus, for the years 1700, 1800, and 1900, a day was dropped, and they were not Leap Years. The year 2000 is exactly divisible by 400, so the day will not be dropped, and February will contain 29 days, making the year 2000 a Leap Year.

The present Gregorian calendar is an improvement over the Julian calendar, but it is still not perfectly accurate. The error is one day in 3000 years.

To convert Julian dates to Gregorian dates, add ten days to the Julian date if the Julian date is from October 5, 1582, through February 28, 1700. Then, for Julian dates from March 1, 1700, through February 28, 1800, add eleven days; add twelve days to Julian dates from March 1, 1800, through February 28, 1900; add thirteen days to Julian dates from March 1, 1900, through February 29, 2000, and so on.

The Gregorian calendar was not universally adopted in 1582. The first countries to put it into use were primarily Roman Catholic nations. The Gregorian calendar was not adopted by most of the Protestant countries until much later. The entire British Empire as well as the American colonies switched in 1752. (September 2, 1752, was followed by September 14, 1752.) An eleven-day adjustment was needed because the Julian calendar had added another day between 1582 and 1752. Sometimes dates that precede the change are referred to as Old Style (or OS). For instance, George Washington's birthday is actually February 11, 1732 (OS). It was only after the switch was made to the Gregorian calendar that his birthday was established as February 22, 1732. Most of the dates in American history have been converted to Gregorian dates, called New Style.

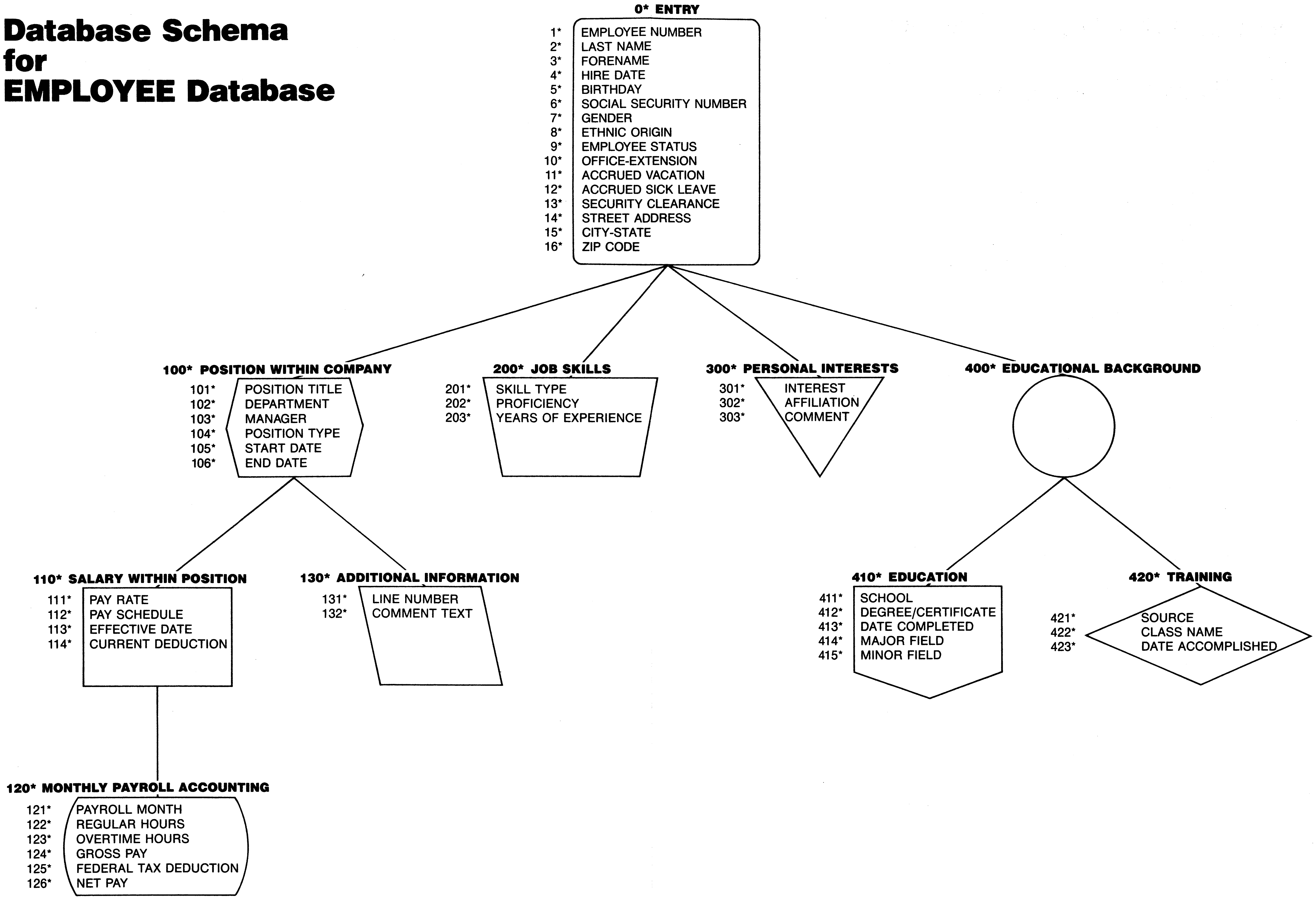
Other countries in the world were even slower to convert to the Gregorian calendar, for instance, Japan in 1873, China in 1912, Greece in 1924, and Turkey in 1927.

Index

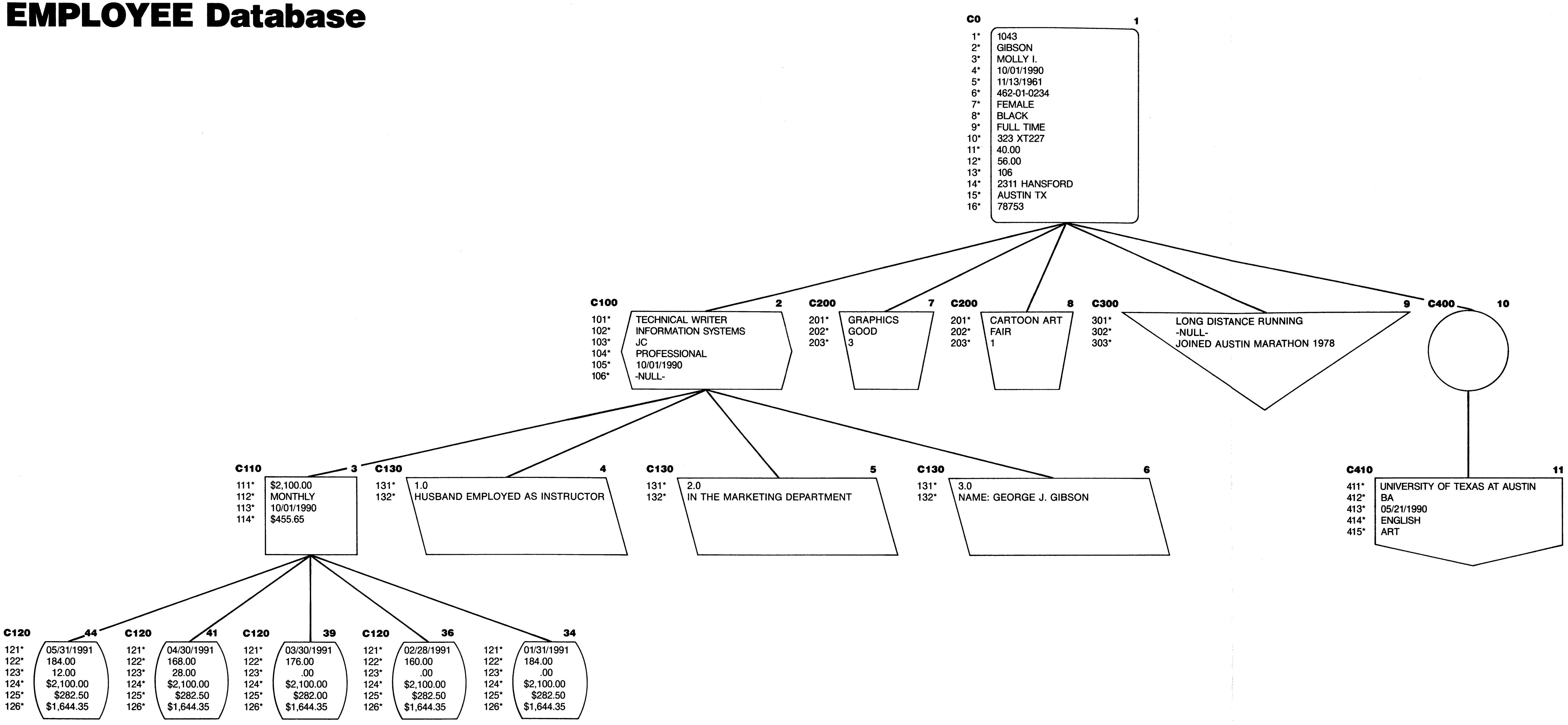
Actual DESCRIBE Output from EMPLOYEE Database

```
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)
```

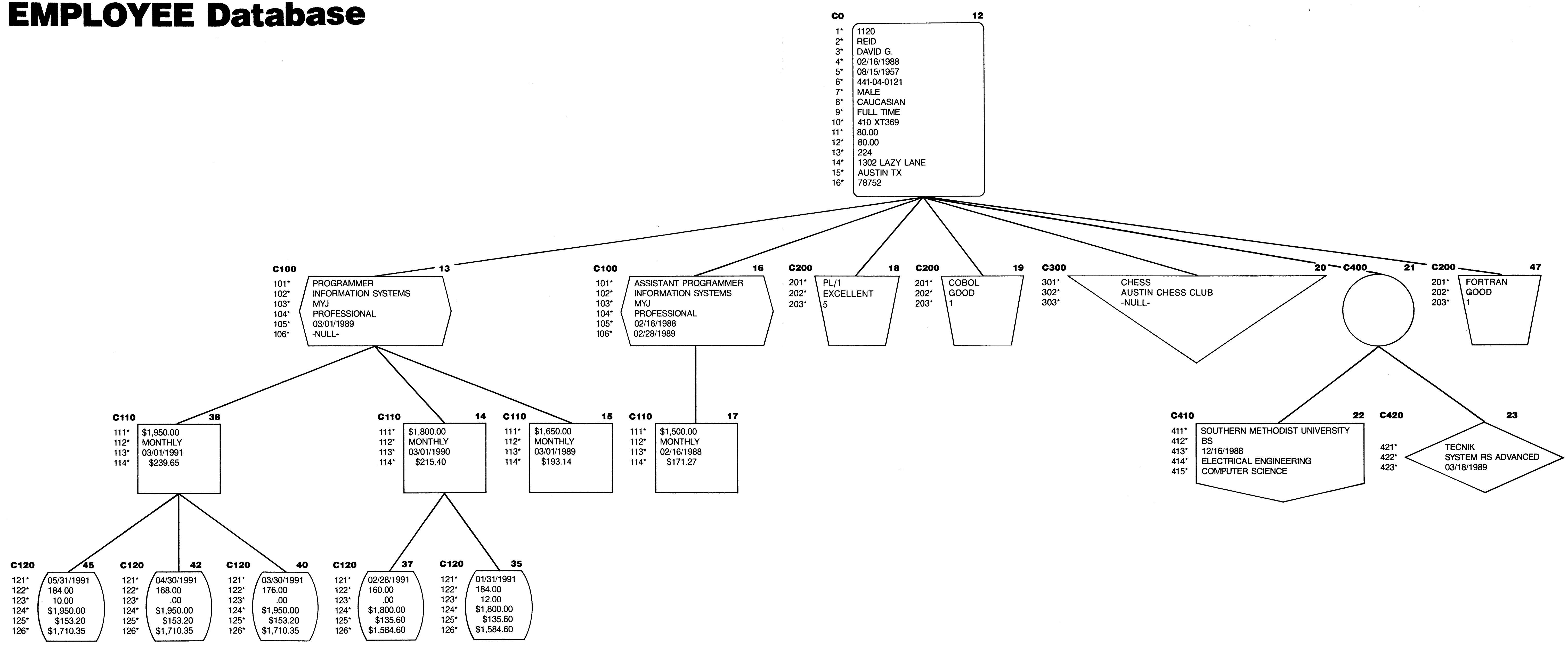
Database Schema for EMPLOYEE Database



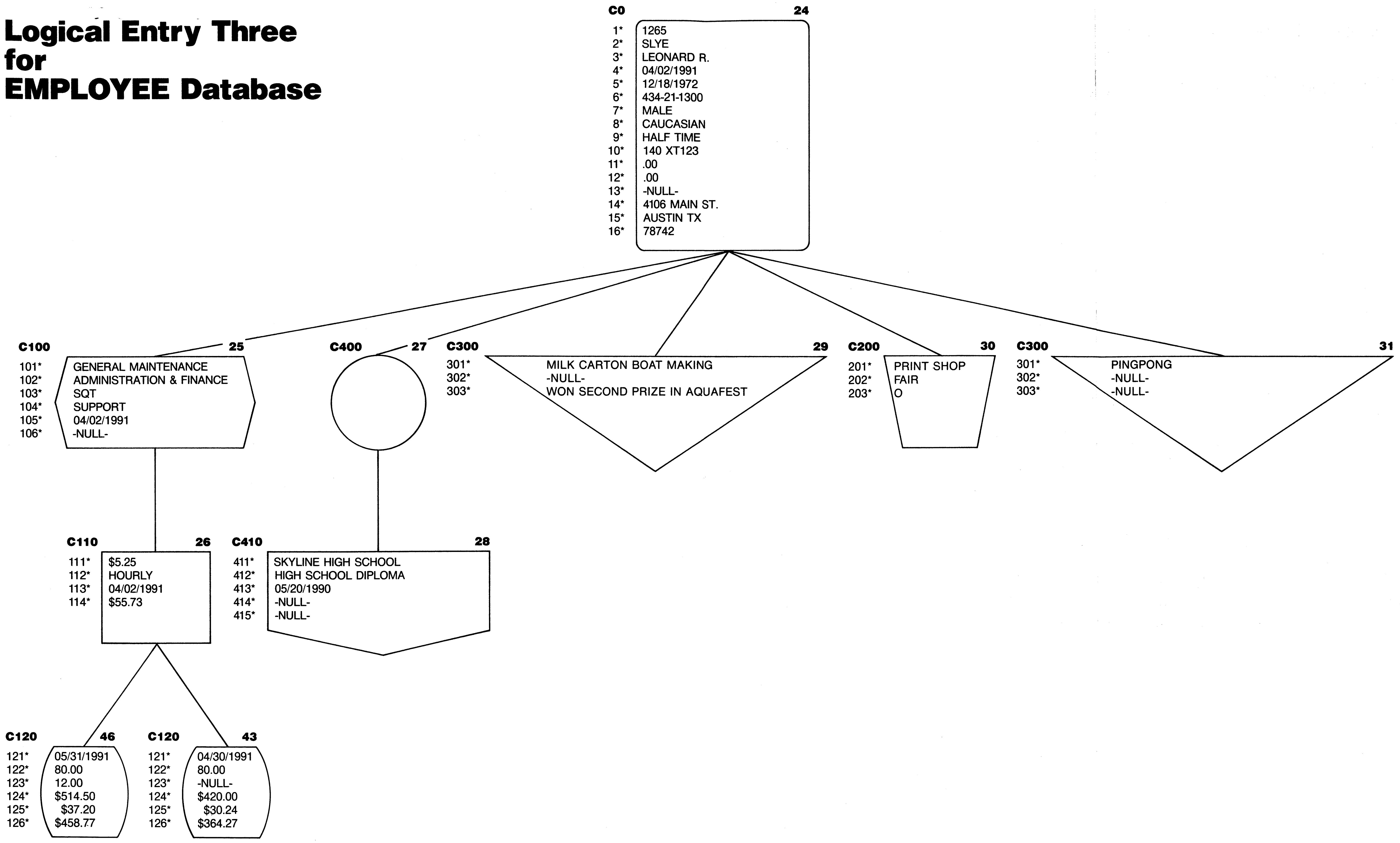
Logical Entry One for EMPLOYEE Database



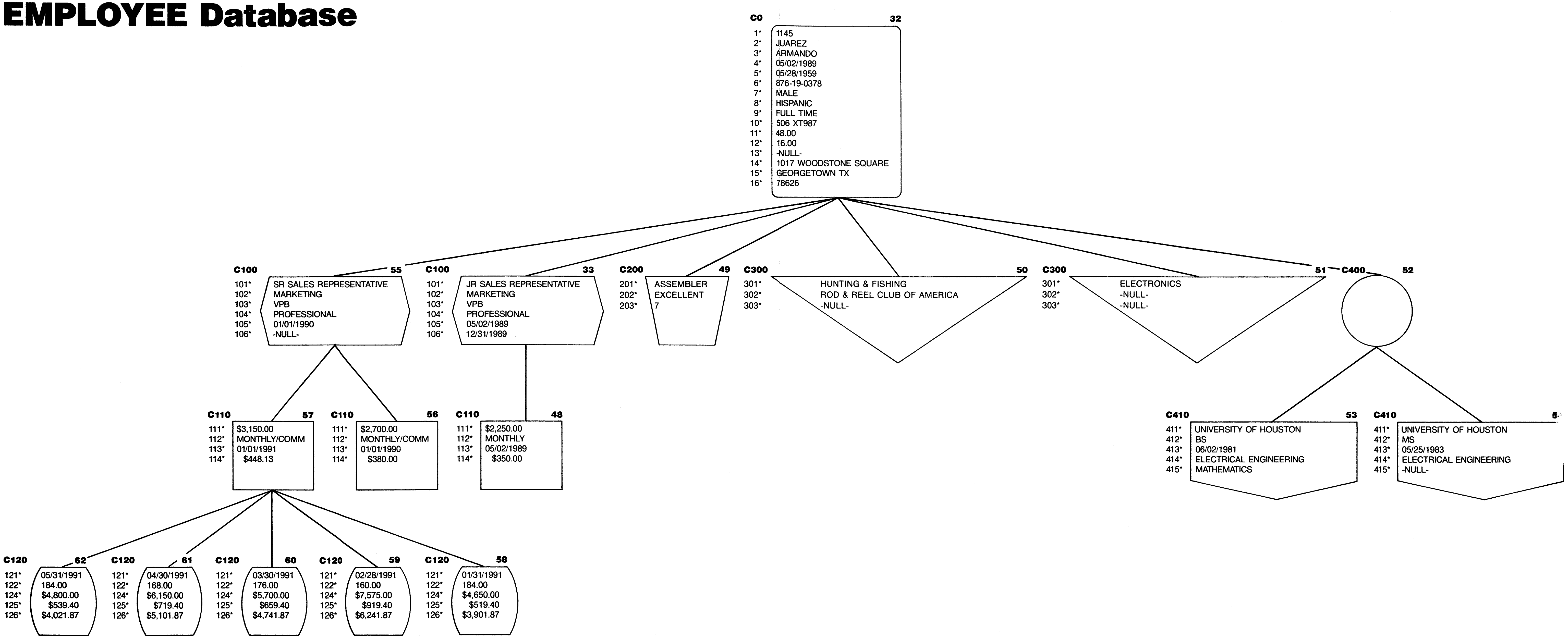
Logical Entry Two for EMPLOYEE Database



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® DEFINE Language, 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.