DMS-170

CYBER DATABASE
CONTROL SYSTEM
VERSION 2
FORTRAN
APPLICATION PROGRAMMING
USER'S GUIDE

CDC® OPERATING SYSTEMS:
NOS 2
NOS/BE 1
## REVISION RECORD

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (03/06/81)</td>
<td>Original release at PSR level 528.</td>
</tr>
<tr>
<td>B (10/08/82)</td>
<td>Updated to reflect FORTRAN Data Base Facility (FDBF) 1.3, CYBER Database Control System (CDCS) 2.3, and use under NOS 2 (but not NOS 1); released at PSR level 564. The guide has been retitled. Major changes include adding documentation of data base transactions, and creating the data base for sample programs with FORTRAN.</td>
</tr>
</tbody>
</table>

Address comments concerning this manual to:

CONTROL DATA CORPORATION  
Publications and Graphics Division  
215 Moffett Park Drive  
Sunnyvale, California 94086

or use Comment Sheet in the back of this manual.
# LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

<table>
<thead>
<tr>
<th>Page</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Cover</td>
<td>-</td>
</tr>
<tr>
<td>Inside Front Cover</td>
<td>B</td>
</tr>
<tr>
<td>Title Page</td>
<td>-</td>
</tr>
<tr>
<td>ii</td>
<td>B</td>
</tr>
<tr>
<td>iii/iv</td>
<td>B</td>
</tr>
<tr>
<td>v</td>
<td>B</td>
</tr>
<tr>
<td>vi</td>
<td>B</td>
</tr>
<tr>
<td>vii</td>
<td>B</td>
</tr>
<tr>
<td>viii</td>
<td>B</td>
</tr>
<tr>
<td>ix</td>
<td>A</td>
</tr>
<tr>
<td>1-1 thru 1-3</td>
<td>A</td>
</tr>
<tr>
<td>1-4 thru 1-7</td>
<td>B</td>
</tr>
<tr>
<td>2-1 thru 2-3</td>
<td>A</td>
</tr>
<tr>
<td>2-4 thru 2-6</td>
<td>B</td>
</tr>
<tr>
<td>3-1 thru 3-4</td>
<td>B</td>
</tr>
<tr>
<td>3-5 thru 3-14</td>
<td>A</td>
</tr>
<tr>
<td>3-15 thru 3-19</td>
<td>B</td>
</tr>
<tr>
<td>4-1 thru 4-6</td>
<td>B</td>
</tr>
<tr>
<td>5-1</td>
<td>B</td>
</tr>
<tr>
<td>5-2</td>
<td>A</td>
</tr>
<tr>
<td>5-3</td>
<td>B</td>
</tr>
<tr>
<td>5-4</td>
<td>A</td>
</tr>
<tr>
<td>5-5</td>
<td>A</td>
</tr>
<tr>
<td>5-6</td>
<td>B</td>
</tr>
<tr>
<td>5-7 thru 5-22</td>
<td>A</td>
</tr>
<tr>
<td>5-23 thru 5-28</td>
<td>B</td>
</tr>
<tr>
<td>6-1 thru 6-3</td>
<td>B</td>
</tr>
<tr>
<td>A-1 thru A-4</td>
<td>A</td>
</tr>
<tr>
<td>B-1 thru B-3</td>
<td>B</td>
</tr>
<tr>
<td>C-1</td>
<td>A</td>
</tr>
<tr>
<td>C-2 thru C-10</td>
<td>A</td>
</tr>
<tr>
<td>C-11 thru C-15</td>
<td>B</td>
</tr>
<tr>
<td>Index-1 thru -3</td>
<td>B</td>
</tr>
<tr>
<td>Comment Sheet</td>
<td>B</td>
</tr>
<tr>
<td>Mailer</td>
<td>-</td>
</tr>
<tr>
<td>Back Cover</td>
<td>-</td>
</tr>
</tbody>
</table>
The DMS-170 data management system clearly defines two roles: the role of a data administrator who develops, controls, and maintains the physical data base; and the role of an application programmer who accesses and manipulates the data within that data base. Although the two roles differ considerably, each role requires a knowledge of the tasks being performed by the other. The data administrator, for example, cannot develop a data base without first understanding what type of applications will be required. The application programmer, on the other hand, cannot successfully access data without first understanding how the data is described and what specific controls have been established.

This guide describes the role of the FORTRAN 5 application programmer who is accessing data within a DMS-170 controlled data base environment. The presence of a data administrator is assumed, and the functions associated with that position are described as they directly affect the application programmer.

You should note that appendix C, entitled The Sample Application, is particularly important. This appendix sets up a working environment complete with stored data for use with sample programs. This environment can be duplicated to provide a better understanding of DMS-170 and the tools that are used to create a total data management system.

The following manuals are of primary interest:

<table>
<thead>
<tr>
<th>Publication</th>
<th>Publication Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN Data Base Facility Version 1 Reference Manual</td>
<td>60482200</td>
</tr>
<tr>
<td>FORTRAN Version 5 Reference Manual</td>
<td>60481300</td>
</tr>
</tbody>
</table>

The following manuals are of secondary interest:

<table>
<thead>
<tr>
<th>Publication</th>
<th>Publication Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYBER Database Control System Version 2 Reference Manual</td>
<td>60481800</td>
</tr>
<tr>
<td>NOS Version 1 Manual Abstracts</td>
<td>84000420</td>
</tr>
<tr>
<td>NOS Version 1 Reference Manual, Volume 1 of 2</td>
<td>60435400</td>
</tr>
<tr>
<td>NOS/BE Version 1 Manual Abstracts</td>
<td>84000470</td>
</tr>
<tr>
<td>NOS/BE Version 1 Reference Manual</td>
<td>60493800</td>
</tr>
<tr>
<td>Software Publications Release History</td>
<td>60481000</td>
</tr>
</tbody>
</table>
CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This manual describes a subset of the features and parameters documented in the FORTRAN Data Base Facility Version 1 Reference Manual. Control Data cannot be responsible for the proper functioning of any features or parameters not documented in the FORTRAN Data Base Facility Version 1 Reference Manual.
CONTENTS

NOTATIONS ix

1. FORTRAN PROGRAMMING WITHIN DMS-170 1-1
   System Components 1-1
      Data Description Language 1-1
         The Schema 1-1
         The Sub-Schema 1-1
       FORTRAN Data Manipulation Language 1-2
       CYBER Database Control System 1-2
          Master Directory 1-2
          CDCS Batch Test Facility 1-2
          Data Base Procedures 1-3
       CYBER Record Manager 1-3
          File Organization 1-3
          Multiple-Index Processing 1-3
   Special Features 1-4
      Concurrency 1-4
      File Privacy 1-4
      Relations 1-4
      Constraints 1-4
      Recovery 1-4
   Summary of DMS-170 Components and Features 1-4

2. ACCESSING THE DATA BASE 2-1
   Interpreting the FORTRAN Sub-Schema 2-1
      FORTRAN Data Manipulation Language 2-1
         DML Language Components 2-1
         Syntax Requirements 2-1
         Statement Positioning 2-1

3. PROCESSING THE DATA 3-1
   Using DML to Access the Data Base 3-1
      Identifying the Sub-Schema 3-1
      Establishing the Interface With CDCS 3-2
      Satisfying Privacy Requirements 3-3
      Opening a Realm 3-3
      Locking/Unlocking a Realm 3-3
      Closing a Realm 3-4
   Using DML to Manipulate Data 3-5
      Writing a Record 3-5
      Reading a Record 3-6
      Sequential Read 3-6
      Random Read 3-6
      Positioning a Realm 3-6
      Rewriting a Record 3-7
      Deleting a Record 3-8
   Using DML to Process Relations 3-8
      Structure of a Relation 3-9
      Using the Sub-Schema 3-10
      Opening a Relation 3-10
      Closing a Relation 3-10
      Reading a Relation 3-12
      Sequential Read 3-12
      Random Relation Read 3-13
      Control Break 3-13
      Null Occurrence 3-13
      Positioning a Relation 3-14
      Updating Realms Joined in a Relation 3-15

4. ERROR PROCESSING AND STATUS HANDLING TECHNIQUES 4-1
   Using ERR and END Processing Options 4-1
   Establishing a Data Base Status Block 4-2
      Error Checking 4-2
      Status Checking 4-2
   Defining Recovery Points 4-4
   Avoiding Constraint Violations 4-4
   Anticipating Deadlock Situations 4-6

5. DEVELOPING FORTRAN PROGRAMS 5-1
   Developing an Application Program 5-1
   Compiling and Executing the Source Program 5-1
   Sample Programs 5-3

6. USING THE CDCS BATCH TEST FACILITY 6-1
   Requirements 6-1
   Obtaining Load Maps 6-1
   Executing the CDCS Batch Test Facility 6-1

APPENDIXES
   A Standard Character Sets
   B Glossary
   C The Sample Application

INDEX

FIGURES

1-1 Schema and Sub-Schema Generation 1-2
1-2 CYBER Record Manager Interface 1-3
2-1 A Basic FORTRAN Sub-Schema 2-2
2-2 A Relational FORTRAN Sub-Schema 2-3
2-3 DML Statement Positioning 2-5
3-1 Sub-Schema AVERAGE 3-2
3-2 Identifying the Sub-Schema 3-2
3-3 Establishing the Interface With CDCS 3-3
3-4 Satisfying Privacy Requirements 3-3
3-5 Opening a Realm 3-4
3-6 Locking/Unlocking a Realm 3-4
3-7 Closing a Realm 3-5
3-8 Terminating the Interface With CDCS 3-5
3-9 Writing a Record 3-5
3-10 Reading Sequentially 3-6
3-11 Reading Randomly 3-6
3-12 Positioning a Realm 3-7
3-13 Rewriting a Record 3-8
3-14 Deleting a Record 3-9
3-15 Tree Structure and Ranks of a Three-Realm Relation 3-9
3-16 Sub-Schema COMPAIR 3-11
3-17 Tree Structure of Record Occurrences 3-13
3-18 Reading a Relation Sequentially 3-13
3-19 Reading a Relation Randomly 3-13
3-20 Null Record Occurrence Examples 3-14
3-21 Positioning a Relation 3-15
4-1 Establishing a Data Base Status Block
4-2 Defining Recovery Points
4-3 Single-File Constraint Example
4-4 Two-File Constraint Example
4-5 Deadlock Processing
5-1 FORTRAN DML Preprocessing
5-2 DML Control Statement
5-3 Executing DML and Compiling the Source Program
5-4 Compiling and Executing the Source Program
5-5 Program RATING
5-6 Program INDAVGE
5-7 Program RELATE
5-8 Program CHARGES

5-9 Program ADMIT
6-1 CDCSBTF Control Statement Format
6-2 Sample FORTRAN Execution of CDCSBTF Batch Test Facility

TABLES
1-1 Summary of DMS-170 Components and Features
2-1 DML Statements
4-1 Error and Status Processing Mechanisms
4-2 Status Block Content
4-3 Locking Operations
6-1 Load Map Switch Settings
The specifications for FORTRAN DML statements and for particular control statements are described in reference formats. The notations used in the reference formats are described as follows:

**UPPERCASE**
Uppercase words are reserved words and must appear exactly as shown. You can use reserved words only as specified in the reference formats.

**Lowercase**
Lowercase words are generic terms that represent user-supplied words or symbols.

[ ] Brackets enclose optional portions of a reference format. You can optionally omit or include all of the format within the brackets.

... Ellipses immediately follow a pair of brackets to indicate that you can optionally repeat the enclosed material.

Punctuation symbols shown within the formats are required unless enclosed in brackets and specifically noted as optional. One or more spaces separate the elements in a reference format. Numbers shown are decimal unless otherwise specified.
DMS-170 is a Control Data software package for data management. The system was designed on the premise that a data base should be centrally controlled and the data within that data base should be completely independent of application programs. In line with this philosophy, the role of data administrator emerged. This individual was to lead the design, programming, implementation, maintenance, and recovery efforts associated with the DMS-170 data management system.

The data administrator is responsible for the structural organization and layout of an entire data base. This individual assigns names to and describes the characteristics of all data items within the data base. This total description is called a schema. The schema is generated by the data administrator and stored as a permanent file.

As a FORTRAN programmer, you probably would never need or even want to access an entire data base. You would, however, need to access selected portions of a data base organized in a number of ways to meet the requirements of your various application programs. The grouping of data base items into separate data base portions is the responsibility of the data administrator. The descriptions of these grouped items are called sub-schemas. Sub-schemas are generated by the data administrator and stored in a permanent file library.

Internal control is handled by the CYBER Database Control System (CDCS). CDCS interprets all data base requests from application programs, ensures the validity of such requests, and passes them along to the input/output processor. The controls exercised by CDCS guarantee that one user cannot alter the contents of the data base and adversely affect another user's program.

The controls designated by the data administrator, incorporated into the schema and sub-schema, and carried out by CDCS relieve application programmers of many tedious tasks such as data description, data conversion, and validity checking.

**SYSTEM COMPONENTS**

The components of DMS-170 that are discussed in this guide include the language that describes the data (Data Description Language); the language that provides data base access to a FORTRAN application program (FORTRAN Data Manipulation Language); the module that controls data base activity (CYBER Database Control System); and the processor that handles all input and output operations (CYBER Record Manager). The components of DMS-170 that are not discussed in this guide include a special, nonprocedural language (Query Update) that provides data base access to programming and nonprogramming users and the COBOL Language extensions that provide data base access to COBOL application programs.

**DATA DESCRIPTION LANGUAGE**

The Data Description Language (DDL) is a compiler language that the data administrator uses to describe data. DDL can generate four types of descriptions: the schema definition that describes an entire data base; the FORTRAN sub-schema definition that describes selected portions of a schema-defined data base for use by a FORTRAN application program; the COBOL sub-schema definition that describes selected portions of a schema-defined data base for use by a COBOL application program; and the QUERY UPDATE sub-schema definition that either describes selected portions of a schema-defined data base or describes an independently controlled data base for use by the interactive query software product Query Update. The data descriptions for the schema and each sub-schema are declared in DDL source statements for input to the DDL compiler.

A block diagram illustrating schema/sub-schema generation is shown in figure 1-1.

**The Schema**

The schema is a detailed description of all the data in a data base. The schema description is generated from DDL statements that name the schema, organize the schema into files (called areas in the schema), describe each record type together with the characteristics of the data in the record, and describe relationships (called relations) and dependency conditions (called constraints) among areas. The schema also includes an access control capability that provides privacy at the area level.

The data administrator writes the DDL source statements and uses them as input to the DDL compiler for compilation into an object schema or schema directory. After storing the directory as a permanent file, the data administrator provides you with pertinent information so you can tailor your FORTRAN program to meet processing requirements. If, for example, you need to access an area that has been defined as having controlled access in the schema, it is the responsibility of the data administrator to supply you with the appropriate privacy key.

**The Sub-Schema**

The sub-schema is a detailed description of selected portions of the data in a data base. The FORTRAN sub-schema description is generated from DDL statements that identify the schema and sub-schema, specify files (called realms in the sub-schema) and the content and structure of records, indicate changes in data format required with pertinent information so you can tailor your FORTRAN program to meet processing requirements.
The data administrator writes the DDL source statements and uses them as input to the DDL compiler for compilation into an object sub-schema or sub-schema directory. After storing the directory in the sub-schema library, the data administrator provides you with a listing of the sub-schema so you can obtain the names and descriptions of the data to be referenced in your FORTRAN program. The data administrator also provides you with the name of the sub-schema library, which you must attach with an operating system ATTACH control statement for DML preprocessing of the Data Manipulation Language statements in your FORTRAN program just before compilation.

FORTRAN DATA MANIPULATION LANGUAGE

The FORTRAN Data Manipulation Language (DML) is the language that provides a FORTRAN application program with access to the DMS-170 controlled data base. The language consists of a series of statements that provide for opening and closing of data base files; reading, writing, updating, and deleting records from those files; and relation processing. The DML statements you include in your FORTRAN source program code are translated by the DML preprocessor into statements acceptable to the FORTRAN compiler.

CYBER DATABASE CONTROL SYSTEM

The central controlling component of DMS-170 is CYBER Database Control System (CDCS), which monitors and interprets all data base requests from application programs. CDCS preprocesses each application program request, performs any necessary data conversion, handles structural differences between the schema and the sub-schema by an operation called mapping, and prepares the request for input/output processing.

Master Directory

The master directory is a file that contains information relating to all data bases, schemas, and sub-schemas known to CDCS. The directory is generated by one of the data base utilities provided through CDCS. The data administrator creates the master directory and stores it as a permanent file. Your application program cannot reference a sub-schema unless information about that sub-schema exists in the master directory. It is the responsibility of the data administrator to ensure the sub-schema is valid. The master directory file is attached through the job stream of CDCS and is automatically available for your job.

CDCS Batch Test Facility

The CDCS Batch Test Facility is an absolute program that you can use during program development and testing. The facility enables you to run CDCS as a normal batch job, which means you can attach a new version of the master directory file each time you run a job.

The program, which resides on the system library, is called into execution by the CDCSBTF control statement. When using this facility, you are responsible for attaching the master directory file and any necessary log files each time you run a job.
Data Base Procedures

Data base procedures are special-purpose subprograms that CDCS calls when specific situations occur during CDCS processing. The data administrator writes the data base procedures and stores them in a permanent file library. The name of the procedure, the point at which it is to be called, and the conditions governing its execution are specified in the schema definition. Loading of data base procedures is handled automatically for you.

CYBER RECORD MANAGER

CYBER Record Manager (CRM) is the processor that performs all input/output operations for FORTRAN as well as the other CYBER host languages operating within DMS-170. The Advanced Access Methods (AAM) file manager handles all operations concerning the physical storage and access of data by application programs. All data base files supported by CDCS are conventional CRM files.

All necessary information regarding the characteristics of a data base file is supplied to CRM at schema compilation time. The data administrator specifies appropriate parameters on FILE control statements that are included in the DDL source deck when the schema is created. In DMS-170, all communication with CRM is handled automatically for you.

A block diagram illustrating the CRM interface with CDCS and the data base is shown in figure 1-2.

File Organization

File organization information is stored in the schema directory. The three file organizations allowed for data base files that are to be accessed through CDCS are: indexed sequential, direct access, and actual key.

Records in indexed sequential files are stored in ascending order by key. An application program can access the records either randomly by key or sequentially.

Records in direct access files are stored randomly in fixed-length blocks. The number of the block to receive a record is determined by a calculation performed by the system on the record. An application program can access the records either randomly by key or sequentially.

Records in actual key files have key values assigned by the system. The key value is a number that identifies the block and the position within the block in which the record is stored. An application program can access the records either randomly by actual key or sequentially.

The primary key is specified in the schema. A listing of the sub-schema provides you with this information.

Multiple-Index Processing

Multiple-index processing is performed when alternate keys are defined for a file. An index is created for each alternate key in a data file when the file is created. The indexes are updated automatically whenever the data file is updated. An application program can retrieve the records by the primary key or by an alternate key.

Each alternate key is specified in the schema. A listing of the sub-schema provides you with this information.

---

Figure 1-2. CYBER Record Manager Interface
SPECIAL FEATURES

Eight special features with which you need to be familiar are: concurrency, immediate return, file privacy, relations, constraints, data base versions, recovery, and data base transactions. When these mechanisms are present in the CDCS operating environment, some action on the part of your application program might be required.

CONCURRENCY

The concurrency feature allows two or more application programs to access the same data base file at the same time for retrieval or update purposes. During concurrent update operations, CDCS provides a locking mechanism by which files and records can be locked and unlocked at appropriate times.

CDCS always locks the current record whenever the file is opened for input/output. Your application program, however, can issue explicit lock and unlock requests for CDCS to lock the entire file. By issuing a lock request, your program prevents other jobs from updating the file that it is using until it issues an unlock request. The file being locked and unlocked must be a file identified in the sub-schema.

A deadlock situation can occur when a program attempts to access files or records that have been locked by CDCS for other programs. When this situation occurs, CDCS arbitrarily releases the locked resources held by one of the contending programs. To ensure proper recovery handling in this type of situation, you should include appropriate code in your FORTRAN program.

IMMEDIATE RETURN

The immediate return feature of CDCS provides FORTRAN application programs with the ability to receive an immediate response from CDCS when either CDCS cannot get the resources it needs, or a fatal error occurs. When this feature is used, CDCS returns control to the application program.

The immediate return feature cannot be enabled before CDCS is invoked. See the CDCS 2 Application Programming reference manual for more information.

FILE PRIVACY

The file privacy feature provides file access control. When file privacy has been specified in the schema, your program must supply privacy keys to gain access to the file.

The data administrator provides you with this information so you can ensure your FORTRAN program meets the privacy requirements when CDCS checks for appropriate privacy keys.

RELATIONS

The relational data base feature allows files to be linked together into a logical relationship called a relation. An application program can access the data from related files with a single read request. Relations are specified in the schema. Any relation that is available to an application program is specified in the sub-schema.

Your application program can access a relation by specifying a single read request with the name of the relation that is to be read. CDCS processes the request and returns a record occurrence from each file in the relation to your program's working storage area for the file.

The data administrator can place limitations on relations by including restrictions in the sub-schema. Restrictions are in the form of qualification criteria that must be satisfied before a record occurrence is made available to your program.

A listing of the sub-schema provides you with the name of the relation and indicates what specific restrictions apply.

CONSTRAINTS

The constraint feature allows controls to be imposed on update operations involving logically associated files. Constraints protect the integrity of the data base by allowing update operations to be performed only when specific conditions are satisfied. Constraints are specified in the schema and are enforced by CDCS.

The data administrator provides you with information concerning constraints. You can avoid constraint violations by becoming familiar with the rules that apply when modifying files on which constraints have been imposed.

DATABASE VERSIONS

The data base version feature of CDCS allows an application program to use the same schema and subschema to access more than one group of permanent files corresponding to the areas in the schema; each of these groups is defined as a data base version. Data base versions are defined by the data administrator in the master directory. By specifying use of different versions, an application program can perform operations on different groups of files, each group forming a data base version. For detailed information about this feature, see the CDCS 2 Application Programming reference manual.

RECOVERY

The recovery feature provides for reconstruction of a damaged or inconsistent data base and provides for the removing of updates made with erroneous logic. The data base can be recovered when physical storage or system failure occurs and all or part of the data base is lost or otherwise unreadable. The data base can be restored to a previous checkpoint or beginning of job when an application program failure or logic error occurs.
Recovery operations are made possible through a logging facility, which is the recording of user interactions with a data base file. Logging requirements are defined by the data administrator for a schema and serviced by CDCS. CDCS records the logging information on an independent file that ultimately serves as input for data base recover and restore operations. Log files, if specified, are attached through the job stream of CDCS and are automatically available to record the interactions of your program with the data base.

**DATA BASE TRANSACTION**

The data base transaction feature of CDCS provides the FORTRAN application program with the ability to group a series of data base updates into a logical unit, called a data base transaction.

The application program specifies the beginning of the transaction, performs the update operations, and specifies the end of the transaction, which can be either a commit or a drop. When the application program specifies a commit operation, all updates made within the transaction become permanent. When the application program specifies a drop operation, all updates within the transaction are reversed; therefore, the data base remains in the state it was in before the beginning of the transaction.

If the application program fails to commit a transaction because of system or program failure, automatic recovery is performed; that is, the transaction is dropped and the data base is restored to its state before the beginning of the transaction.

Transaction processing also provides an application program with the ability to determine the point at which to restart processing after a system failure. The application program can use this feature to determine the last transaction that was committed before the system failure occurred. The program can then determine the point at which processing should be restarted.

**SUMMARY OF DMS-170 COMPONENTS AND FEATURES**

A summary of DMS-170 components and features appears in table 1-1. This table provides a quick reference for appropriate information.

<table>
<thead>
<tr>
<th>Component/Feature</th>
<th>Definition</th>
<th>Information Appears In</th>
<th>Programmer Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate key</td>
<td>A key other than the primary key by which a file can be accessed; defined by the data administrator.</td>
<td>Sub-schema listing</td>
<td>On a random read, set the key to a value indicating the desired record occurrence.</td>
</tr>
<tr>
<td>CDCS Batch Test Facility</td>
<td>A non-concurrent version of CDCS for use during program development.</td>
<td>N/A</td>
<td>Attach the master directory when executing the application program.</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Simultaneous access to the same data by two or more application programs.</td>
<td>N/A</td>
<td>Include appropriate code in the application program to handle a deadlock situation; deadlock can occur when two programs are contending for access to a locked file or record.</td>
</tr>
<tr>
<td>Constraints</td>
<td>Controls imposed on records in associated files or on items in a single file to protect the integrity of the data base during update operations; defined by the data administrator.</td>
<td>Schema</td>
<td>Obtain information from the data administrator. Follow the rules for modifying files on which constraints have been imposed.</td>
</tr>
<tr>
<td>CYBER Database Control System (CDCS)</td>
<td>The central controlling module of DMS-170.</td>
<td>N/A</td>
<td>None.</td>
</tr>
<tr>
<td>CYBER Record Manager (CRM)</td>
<td>The input/output processor for DMS-170 operations.</td>
<td>N/A</td>
<td>None.</td>
</tr>
<tr>
<td>Data base procedures</td>
<td>Special-purpose routines that perform predefined operations; written by the data administrator.</td>
<td>Schema</td>
<td>None.</td>
</tr>
<tr>
<td>Component/Feature</td>
<td>Definition</td>
<td>Information Appears In</td>
<td>Programmer Action</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Data base transaction</td>
<td>A series of update operations identified by a user-assigned transaction identifier. A transaction is bracketed by a begin transaction operation and either a commit or drop operation.</td>
<td>N/A</td>
<td>Include appropriate code in the application program to bracket the series of update operations and to assign transaction identifiers. Transaction log files must have been defined by the data administrator in the master directory.</td>
</tr>
<tr>
<td>Data base status block</td>
<td>An array defined within an application program to which CDCS returns information concerning the status of operations on data base files and relations.</td>
<td>N/A</td>
<td>Include appropriate code in the application program.</td>
</tr>
<tr>
<td>Data base version</td>
<td>A set of permanent files that is associated with the areas described by the schema; defined by the data administrator.</td>
<td>Master directory</td>
<td>To use a data base version other than MASTER (which is otherwise assumed), specify the version name in the application program.</td>
</tr>
<tr>
<td>Data Description Language (DDL)</td>
<td>The language that is used to structure a schema and sub-schema; used by the data administrator.</td>
<td>N/A</td>
<td>None.</td>
</tr>
<tr>
<td>File organization</td>
<td>The predetermined arrangement of stored data; indexed sequential, direct access, or actual key; defined by the data administrator.</td>
<td>Schema</td>
<td>None.</td>
</tr>
<tr>
<td>File privacy</td>
<td>A situation in which an application program can only gain access to a file by supplying a privacy key; defined by the data administrator.</td>
<td>Schema</td>
<td>Obtain information from the data administrator. Include a PRIVACY statement in the application program.</td>
</tr>
<tr>
<td>FORTRAN Data Manipulation Language (DML)</td>
<td>The language that provides a FORTRAN application program with access to the DMS-170 controlled data base.</td>
<td>N/A</td>
<td>Include appropriate DML statements in the FORTRAN source program code.</td>
</tr>
<tr>
<td>Immediate return</td>
<td>A feature of CDCS that provides FORTRAN application programs with the ability to receive immediate response from CDCS when either CDCS cannot get the resources it needs, or a fatal error occurs.</td>
<td>N/A</td>
<td>Include appropriate code in the application program.</td>
</tr>
<tr>
<td>Log files</td>
<td>Disk files on which user interactions with data base files are recorded for recovery purposes.</td>
<td>Master directory</td>
<td>None.</td>
</tr>
<tr>
<td>Master directory</td>
<td>A file containing information relating to all data bases, schemas, and sub-schemas known to CDCS; created by the data administrator.</td>
<td>N/A</td>
<td>Attach the master directory only when executing the program through the CDCS Batch Test Facility.</td>
</tr>
<tr>
<td>Multiple-index processor</td>
<td>A processor that allows CRM files to be accessed by alternate keys.</td>
<td>N/A</td>
<td>None.</td>
</tr>
<tr>
<td>Component/Feature</td>
<td>Definition</td>
<td>Information Appears In</td>
<td>Programmer Action</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td><strong>Primary key</strong></td>
<td>A key that must be defined for a file when the file is first created; defined by the data administrator.</td>
<td>Sub-schema listing</td>
<td>On a random read, set the key to a value indicating the desired record occurrence.</td>
</tr>
<tr>
<td><strong>Recovery:</strong></td>
<td>A means by which a data base can be automatically recovered in case of a system or program failure.</td>
<td>N/A</td>
<td>Data base transactions provide for automatic recovery; use them for sensitive updates.</td>
</tr>
<tr>
<td>Automatic</td>
<td>N/A</td>
<td>Data base transactions provide for automatic recovery; use them for sensitive updates.</td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>N/A</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td><strong>Relations</strong></td>
<td>Logical structures formed by the joining of files; permit retrieval of data from more than one file at the same time; defined by the data administrator.</td>
<td>Sub-schema listing</td>
<td>To read a relation, specify a single read request with the name of the relation. During update, follow the rules for updating files joined in a relation.</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>Criteria that must be satisfied in a relation before a record occurrence can be made available to the application program; defined by the data administrator.</td>
<td>Sub-schema listing</td>
<td>None.</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>A detailed description of all the data in a data base; created by the data administrator through DDL.</td>
<td>Schema listing</td>
<td>None.</td>
</tr>
<tr>
<td><strong>Sub-schema</strong></td>
<td>A detailed description of selected portions of the data in a data base; created by the data administrator through DDL.</td>
<td>Sub-schema listing</td>
<td>Attach the sub-schema library in which the sub-schema resides for DML preprocessing of the application program.</td>
</tr>
</tbody>
</table>
Every DMS-170 data base file that is to be accessed through FORTRAN must be described in a directory called a FORTRAN sub-schema. The data administrator, working with application programmers, is responsible for creating the sub-schemas. Every FORTRAN program that accesses a DMS-170 data base file must use the FORTRAN Data Manipulation Language (DML). The application programmer is responsible for coding appropriate DML statements and including them in the FORTRAN source program.

This section details the two principal data base access tools: the sub-schema that describes the data, and the language that provides access to that data.

INTERPRETING THE FORTRAN SUB-SCHHEMA

The data administrator tailors sub-schemas to meet specific applications. Assume, for example, you have an application that requires access to only two fields in a data base file: student IDs and tuition charges. The data administrator might provide you with a sub-schema that resembles sub-schema SAMPLE1 shown in figure 2-1.

With the exception of the sub-schema library name and any required privacy keys, the listing provides you with complete information. The handwritten notation in this example indicates the sub-schema library name is DDLLIB. If you were planning to compile an application program using sub-schema SAMPLE1, you would need to attach DDLLIB. Since no privacy key is required, the schema obviously imposes no access control on realm ACCOUNT.

Notice the three aliases assigned. Since symbolic names in FORTRAN cannot exceed seven characters and cannot include a hyphen, the data administrator changes the names for your application.

Assume, for example, you have an application that requires access to two data base files. Assume also, that you need a relationship between the two files so that you can search one file and retrieve corresponding records from the other. The data administrator might provide you with a sub-schema that resembles sub-schema SAMPLE2 in figure 2-2.

The handwritten notation in this example indicates the sub-schema library name is DDLLIB. If you were planning to compile an application program using sub-schema SAMPLE2, you would need to attach DDLLIB. The schema apparently imposes access control on realm FILE2. The privacy key XX99 must be included in a DML PRIVACY statement to gain access to that realm.

FORTRAN DATA MANIPULATION LANGUAGE

The FORTRAN Data Manipulation Language (DML) is the means through which your FORTRAN program accesses the data base. You must code the DML statements along with FORTRAN statements in the FORTRAN source program. The DML statements identify the sub-schema, establish an interface with CDCS, and provide access to realms defined by the sub-schema. DML statements can appear both in the main program and in subprograms.

The DML statements are translated into statements acceptable to the FORTRAN compiler. The DML preprocessor performs the translation and writes the translated statements to a file along with the FORTRAN statements in the source program. This new file is then the input file to the FORTRAN compiler.

DML LANGUAGE COMPONENTS

The DML language components include DML statement keywords, recognized symbols and punctuation, and user-supplied names of variables and constants. These components are grouped together into statements for input to the DML preprocessor, which translates each statement appropriately into a FORTRAN specification or CALL statement.

The first word of a statement is always a DML keyword that identifies the task to be performed. Most keywords are followed by user-supplied elements and sometimes are followed by additional keywords.

A list of available DML statements is shown in table 2-1 for reference purposes. The statements are listed in alphabetic order by the leading word (keyword), which identifies the purpose of the complete statement. The comments column provides specific rules for the statement and includes applicable default options.

SYNTAX REQUIREMENTS

The syntax requirements and coding conventions for DML statements are exactly the same as for FORTRAN statements. The following restrictions apply:

- A DML statement cannot be the object of a logical IF.
- A DML statement must not reference files that are referenced elsewhere in the program by a conventional FORTRAN input/output statement or by a FORTRAN PROGRAM statement.

Any executable DML statement can have a statement label. The DML preprocessor copies the label into the translated FORTRAN statement.

STATEMENT POSITIONING

Some DML statements require special positioning within the FORTRAN source program. These requirements are illustrated in figure 2-3.
Figure 2-1. A Basic FORTRAN Sub-Schema
The subschema name is SAMPLE2; the schema name is UNIVERSITY.

New names are assigned to two realms, records, and items. The new names are FILE1, FIL1REC, and FILE1ID, respectively, for the first realm; FILE2, FIL2REC, and FILE2ID, respectively, for the second realm.

The subschema provides access to two realms: FILE1 and FILE2.

The subschema provides access to record FIL1REC within FILE1.

Two items in record FIL1REC can be accessed: FILE1ID and MAJOR. FILE1ID is described as an 11-character item; MAJOR is described as a 20-character item.

The subschema provides access to record FIL2REC within FILE2.

Three items in record FIL2REC can be accessed: IDENT, which is described as a 14-character item; FILE2ID, which is described as an 11-character item; and GRADE, which is described as a real item.

A relation called REL1 is available. FILE1 and FILE2 are joined.

Primary and alternate keys are listed.

A restriction is placed on record FIL2REC. Only records with a grade point average of 4.0 are to be returned.

Library name DDLIB.

Primary key XX99 needed for FILE2.

Figure 2-2. A Relational FORTRAN Sub-Schema
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNID</td>
<td>Obtains a restart identifier assigned by CDCS.</td>
<td>The program must execute the ASSIGNID statement before processing data base transactions in order to determine the status of a data base transaction in a restart operation after a system failure occurs. The FINDTRAN statement is used in the restart operation.</td>
</tr>
<tr>
<td>BEGINTRAN</td>
<td>Begins processing of a data base transaction.</td>
<td>Updates are considered temporary until the data base transaction is committed.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Ends processing of a realm.</td>
<td>Realms can be opened and closed any number of times by a program.</td>
</tr>
<tr>
<td>CLOSE relation</td>
<td>Ends processing of the realms joined in a relation.</td>
<td>Relations can be opened and closed any number of times by a program.</td>
</tr>
<tr>
<td>COMMITTRAN</td>
<td>Completes processing of a data base transaction.</td>
<td>This statement causes all updates within the data base transaction to be considered permanent.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Removes a record from a realm.</td>
<td>The record being deleted is the record most recently read from the realm. The value of the primary key cannot change after the last read.</td>
</tr>
<tr>
<td>DROPTRAN</td>
<td>Cancels processing of a data base transaction.</td>
<td>All realms are restored to the states that existed just before the data base transaction began.</td>
</tr>
<tr>
<td>FINDTRAN</td>
<td>Obtains information for a program restart operation after system failure.</td>
<td>The program must have obtained a restart identifier by executing the ASSIGNID statement in order to use the FINDTRAN statement.</td>
</tr>
<tr>
<td>INVOKE</td>
<td>Establishes the interface between the executing program and CDCS.</td>
<td>The statement must be executed before any other DML statements except SUBSCHEMA.</td>
</tr>
<tr>
<td>LOCK</td>
<td>Establishes an exclusive or protected lock on realms. Exclusive prohibits read and update operations on the realm. Protected prohibits only update operations (allows read operations).</td>
<td>The realm must be a realm described in the sub-schema.</td>
</tr>
<tr>
<td>NEWVERSION</td>
<td>Changes the data base version being used by an application program.</td>
<td>All sub-schema realms must be closed before executing the NEWVERSION statement. See the CDCS 2 Application Programming reference manual for details.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Initiates processing of a realm.</td>
<td>If the processing mode is not specified, the realm is opened for input/output.</td>
</tr>
<tr>
<td>OPEN relation</td>
<td>Initiates processing of the realms joined in a relation.</td>
<td>If the processing mode is not specified, the relation is opened for input/output. A processing mode of open for output only is not valid.</td>
</tr>
<tr>
<td>PRIVACY</td>
<td>Establishes the right of a program to access a realm.</td>
<td>If the processing mode is not specified, the realm can be accessed for input/output. The mode must be the same as the mode indicated in the OPEN statement.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>READ</td>
<td>Transfers data from a realm record to the variables defined in the sub-schema record description.</td>
<td>If a key is not specified, the read is sequential. If a key is specified, the value of the referenced key must be set by the program before the read is executed.</td>
</tr>
<tr>
<td>READ relation</td>
<td>Transfers data from the relation records to the corresponding variables defined in the sub-schema record descriptions.</td>
<td>If a key is not specified, the read is sequential. If a key is specified, the key must be in the root realm; the value of the referenced key must be set by the program before the read is executed.</td>
</tr>
<tr>
<td>REWRITE</td>
<td>Replaces the last record read with a new record, using the current values of the variables defined in the sub-schema record description.</td>
<td>The value of the primary key must not have changed since the last read.</td>
</tr>
<tr>
<td>START</td>
<td>Logically positions a realm for a subsequent sequential read operation.</td>
<td>The processing mode must be either input or input/output. If a key is specified, it must be a primary or alternate key defined for the realm. If a key is not specified, positioning is by primary key.</td>
</tr>
<tr>
<td>START relation</td>
<td>Logically positions the root realm (the first realm named in the sub-schema) of a relation for a subsequent relation read operation.</td>
<td>The processing mode must be either input or input/output. If a key is specified, it must be a primary or alternate key that is defined in the root realm. If a key is not specified, positioning is by primary key of the root realm.</td>
</tr>
<tr>
<td>SUBSCHEMA</td>
<td>Identifies the sub-schema to be used by the program.</td>
<td>A FORTRAN program can reference only one sub-schema.</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>Terminates the interface between the FORTRAN program and CDCS.</td>
<td>When a TERMINATE statement is issued, an INVOKE statement must be executed before any other DML statements are issued.</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>Releases a lock on a specified realm and releases all record locks.</td>
<td>The realm must be a realm described in the sub-schema.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Writes a record, using the current values of the variables defined in the sub-schema record description.</td>
<td>Schema record data items that are not defined in the sub-schema are given null values by CDCS.</td>
</tr>
</tbody>
</table>
Figure 2-3. DML Statement Positioning
Processing data base files within the DMS-170 environment involves several steps. These steps are:

1. Obtain a current listing of the sub-schema from the data administrator so you can have the names and descriptions of the data your program will be referencing.

2. Obtain the name of the appropriate sub-schema library from the data administrator. You will need to attach this library for preprocessing your program.

3. Ask the data administrator if any realms in the sub-schema are defined in the schema as having controlled access. When access is controlled, you must know the privacy key.

4. Ask the data administrator if any constraints exist in the schema. When constraints exist, CDCS enforces them by not allowing updates that violate constraints.

5. Code the FORTRAN program and include appropriate Data Manipulation Language (DML) statements for opening, closing, and processing sub-schema realms.

6. Preprocess and compile the FORTRAN program. Include, in the job stream before the FTN5 control statement, an ATTACH control statement naming the sub-schema library and a DML control statement to execute the DML preprocessor.

7. When compilation is successful, execute the FORTRAN program. Include an LDSET control statement to load the DMS-170 library.

DML statements are available to perform a variety of operations on data base items described in a FORTRAN sub-schema. This section describes these statements and presents them in the following sequence:

Data Base Access

| SUBSCHEMA | INVOCER |
| PRIVACY |
| OPEN |
| LOCK/UNLOCK |
| CLOSE |
| TERMINATE |

Data Base Manipulation

| WRITE |
| READ |
| START |
| REWRITE |
| DELETE |

Relation Access

| OPEN |
| CLOSE |
| READ |
| START |

For purposes of illustration, a new sub-schema named AVERAGE is shown in figure 3-1. This sub-schema is referenced in subsequent examples. The examples show portions of program MODEL which illustrate statements necessary for particular data base operations.

The sub-schema provides the following information:

- The realm (file) to be accessed is named CFILE.
- The record is named CRECORD.
- A character item named IDENT is the primary key.
- A character item named STUDENT is an alternate key.
- A character item named COURSE is an alternate key.
- A real item named GRADE is an alternate key.

The handwritten notation on the listing indicates the sub-schema is stored on a library named SSLIB. The notation also indicates CFILE has controlled access and requires a privacy key of XX99.

USING DML TO ACCESS THE DATA BASE

To access a data base, a FORTRAN program must identify the sub-schema that the program uses, establish an interface with CDCS, satisfy privacy requirements, and perform the usual functions of opening and closing files. The following paragraphs describe these functions and the DML statements that you include in your program to provide these functions.

IDENTIFYING THE SUB-SCHEMA

To identify the sub-schema, you must include a SUBSCHEMA statement in your program. This must be the first DML statement to appear in your program. The format is:

```
SUBSCHEMA(sub-schema-name)
```

The SUBSCHEMA statement is required. You must position the statement in the program as follows:

- After the specification statements
- Before the first DATA or NAMELIST statement
- Before any statement function
- Before any executable statement

At the point where the DML preprocessor encounters the SUBSCHEMA statement, the DML preprocessor copies into the source program the text declaration and DATA statements resulting from the sub-schema.
In this way, DML provides your program with the ability to reference all records, data items, and relations that are described in the sub-schema.

In a program using the sample sub-schema AVERAGE, the SUBSCHEMA statement appears as shown in figure 3-2.

**ESTABLISHING THE INTERFACE WITH CDCS**

To establish the interface with CDCS, you must include an INVOKE statement in your program. This must be the second DML statement to appear in your program. The format is:

```
INVOKE
```

The INVOKE statement is required. The statement must appear in the program before any other DML statement except SUBSCHEMA.

When the INVOKE statement is executed, CDCS automatically attaches for use by the program all realms described in the sub-schema identified by the program.

In a program using the sample sub-schema AVERAGE, the INVOKE statement appears as shown in figure 3-3.

**SATISFYING PRIVACY REQUIREMENTS**

If a realm is defined in the schema as having controlled access, your program must provide a privacy key to access the realm. To provide the privacy key, you must include the PRIVACY statement in your program. The format is:

```
PRIVACY
```

**Library name SSLIB.**

Privacy key XX99 needed for CFIL.
Figure 3-3. Establishing the Interface With CDCS

PRIVACY(realm-name, MODE=mode, PRIVACY=privacy key)

where

mode = I (access allowed for input)
       10 (access allowed for both input and output, called input/output; default)
       0 (access allowed for output)

privacy key = character constant, variable name, unsubscripted array name

A privacy key can be from 1 through 30 characters in length. If you use a character constant to specify a privacy key, enclose the character string in apostrophes. If you use a variable name to specify a privacy key, define the variable as type CHARACTER*30. Ensure that the privacy key is left-justified and blank filled in the field of the variable. If you use an array name to specify the privacy key, define the array as a 3-word array. Ensure that the privacy key is left-justified and blank filled in the field of the array.

The PRIVACY statement is required when access is controlled. A separate PRIVACY statement is required for each realm defined with controlled access. The PRIVACY statement must be executed before the statement that opens the realm.

The handwritten notation on the sample sub-schema listing (figure 3-1) indicates access to the realm is controlled and a privacy key called XX99 is required. In a program using the sample sub-schema AVERAGE, the required PRIVACY statement appears as shown in figure 3-4.

OPENING A REALM

Before your program can access any data records in an existing realm, the program must open the realm. To open the realm, you must include the OPEN statement in your program. The format is:

OPEN(realm-name, MODE=mode, [,ERR=s])

For a realm with controlled access, the mode of access indicated in the PRIVACY statement must provide for the access mode indicated in the OPEN statement. For example, if you open a realm for input (MODE=I) you can specify MODE=I0 in the PRIVACY statement.

If a separate privacy key is required for input (MODE=I) and another privacy key is required for output (MODE=0), two PRIVACY statements are required to open a realm for input/output (MODE=I0).

In a program using the sample sub-schema AVERAGE (figure 3-1), the OPEN statement appears as shown in figure 3-5. The MODE option is not included, indicating a default to input/output.
Both the PRIVACY and OPEN statements indicate the same mode, input/output. If an error occurs on open, program execution continues at statement 100.

LOCKING/UNLOCKING A REALM

Whenever your program issues a read request on a realm that is open for input/output, CDCS automatically locks the record that was read (the current record) against update by another user. Through DML, however, your program can prevent other jobs from performing update operations anywhere within a realm by issuing a request that CDCS lock the entire realm. To issue the lock request, you must include the LOCK statement in your program. The format is:

```
LOCK (realm-name [,TYPE=lock-type [,ERR=s3]])
```

where

- `lock-type = character constant or variable`
- `s = label of an executable statement to which control transfers on lock error`

Two types of locking are permitted: exclusive or protected. Exclusive locking prohibits concurrent access to the realm for read or update operations; protected locking allows concurrent access to the realm for read operations but prohibits concurrent update operations.

Lock-type can be specified as either a character constant or a variable. The lock-type option must specify either the value EXCLUSIVE or the value PROTECTED. The lock-type is 9 characters long. If you use a character constant to specify the lock-type, enclose the character string in apostrophes. If you use a variable to specify the lock-type, define the variable as type CHARACTER*9.

The LOCK statement should be executed before any read request with intent to update the record.

CDCS releases the realm lock held for your program when the program issues an unlock request. To issue the unlock request, you must include an UNLOCK statement in your program. The format is:

```
UNLOCK (realm-name [,ERR=s])
```

where

- `s = label of an executable statement to which control transfers on unlock error`

You should judiciously use a realm lock. A realm lock limits other users' access to the realm (file). Additionally, a realm lock overrides the CDCS record locking mechanism, which provides a checking capability on rewriting and deleting records (for additional information, see the paragraphs Rewriting a Record and Deleting a Record).

In a program using the sample sub-schema AVERAGE (figure 3-1), the LOCK and UNLOCK statements appear as shown in figure 3-6. If an error occurs during the lock or unlock process, program execution continues at statement 200 or 300, respectively.

CLOSING A REALM

When your program has completed processing on a realm, the program must close the realm. To close the realm, you must include the CLOSE statement in your program. The format is:

```
CLOSE (realm-name [,ERR=s])
```

where

- `s = label of an executable statement to which control transfers on close error`

Once a program closes a realm, the program can perform no further processing on the realm until it reopens the realm.

In a program using the sample sub-schema AVERAGE (figure 3-1), the CLOSE statement appears as shown in figure 3-7. If an error occurs on close, program execution continues at statement 400.

TERMINATING THE INTERFACE WITH CDCS

To terminate the interface with CDCS, you must include a TERMINATE statement in your program. The format is:

```
TERMINATE
```

Once the TERMINATE statement is executed, no further data base processing can take place without execution of another INVOKE statement.
In a program using the sample sub-schema AVERAGE (figure 3-1), the TERMINATE statement appears as shown in figure 3-8. This statement must be executed before the FORTRAN END or STOP statement.

**PROGRAM MODEL**

SUBSCHEMA(AVERAGE)

INVOKE PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')

OPEN(CFILE,ERR=100)

LOCK(CFILE,ERR=200)

UNLOCK(CFILE,ERR=300)

CLOSE(CFILE,ERR=400)

400 PRINT *, 'ERROR ON CLOSE'

END

Figure 3-8. Terminating the Interface With CDCS

**USING DML TO MANIPULATE DATA**

DML statements are available to create, read, position, and modify data base records. The following paragraphs describe these functions and the DML statements that you must include in your program to provide these functions.

**WRITING A RECORD**

To write a complete record, you must include a WRITE statement in your program. The format is:

```
WRITE(realm-name [,ERR=s])
```

where

```
s = label of an executable statement to which control transfers on write error
```

Before the WRITE statement is executed, the program must set the primary key and all alternate keys to appropriate values. A sub-schema does not always reflect all data items that appear in the schema record; therefore, before allowing the new record to be written to the data base, CDCS gives null values to those schema data items that are not defined in the sub-schema.

In a program using the sample sub-schema AVERAGE (figure 3-1), the WRITE statement appears as shown in figure 3-9. The program sets the primary key IDENT and alternate keys STUDENT, COURSE, and GRADE to appropriate values before the WRITE statement is executed. If an error occurs on write, program execution continues at statement 500.

**PROGRAM MODEL**

SUBSCHEMA(AVERAGE)

INVOKE PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')

IDENT='122-13-6704-09'

STUDENT='122-13-6704'

COURSE='PSY136'

GRADE=3.7

WRITE(CFILE,ERR=500)

500 PRINT *, 'ERROR ON WRITE'

900 CLOSE(CFILE)

TERMINATE

END

Figure 3-9. Writing a Record

This example shows a program writing a record to an existing file, CFILE. Before the program writes to this file, the PRIVACY and OPEN statements establish access for input/output (MODE=IO). If the program were creating this file, the OPEN and PRIVACY statements would have to establish access for output only (MODE=O).
READING A RECORD

To read a record, you must include a READ statement in your program. The format is:

```
READ(realm-name C,KEY symbol item-name} {C,ERR=s} {C,END=s})
```

where

- `symbol` = `.EQ. .GT. .GE.`
- `item-name` = primary or alternate key
- `s` = label of an executable statement to which control transfers on read error (ERR)
- `label of an executable statement to which control transfers on end-of-file (END)`

When you omit the KEY option, the read operation is sequential. When you include the KEY option, the read operation is random.

The END option is valid only for a sequential read operation.

Sequential Read

A sequential read accesses the record occurrence located at the current record position. Successive read operations return record occurrences by position. Indexed sequential files are sequenced in ascending primary key order, actual key files are sequenced by block and record slot within the block, and direct access files are sequenced by position in home blocks.

Typical FORTRAN 5 statements issued outside of a data base environment terminate with a fatal error if EOF is sensed and a test for EOF status is not included in the FORTRAN READ statement. If EOF status is not tested in DML, program execution continues with the next statement. Consequently, it is necessary to test for EOF on a DML sequential read operation. You can handle this test in one of two ways:

- Include the END option on the READ statement. When EOF is sensed, program execution continues at the statement specified in the option.
- Test for an EOF value of 100g in the data base status block. This option is described in section 4.

In a program using the sample sub-schema AVERAGE (figure 3-1), if an error occurs on read, program execution continues at statement 600. The READ statement includes the END option to test for EOF. When EOF is reached, program execution continues at statement 900.

Random Read

A random read accesses a record occurrence by the value of a referenced primary or alternate key. The program must set the value of the referenced key before the READ statement is executed.

Figure 3-10. Reading Sequentially

In a program using the sample sub-schema AVERAGE (figure 3-1), a random read appears as shown in figure 3-11. The program sets the alternate key GRADE to the value 4.0. This read returns from CF1E the first record occurrence in which the alternate key GRADE has the value 4.0. If an error occurs on read, program execution continues at statement 600.

Figure 3-11. Reading Randomly

POSITIONING A REALM

To position a realm for subsequent sequential read operations, you must include a START statement in your program. The format is:

```
START(realm-name C,KEY symbol item-name] {C,ERR=s])
```

where

- `symbol` = `.EQ. .GT. .GE.`
- `item-name` = primary or alternate key
- `s` = label of an executable statement to which control transfers on start error

Before the START statement is executed, the program must have opened the realm for input or input/output.
When you omit the KEY option, the realm is positioned by primary key value; the realm is positioned to the record occurrence with a primary key value equal to the current value of the primary key item. When you include the KEY option, the realm is positioned to the first record occurrence with a matching key value.

In a program using the sample sub-schema AVERAGE (figure 3-1), both forms of the START statement appear as shown in figure 3-12. If an error occurs on start, program execution continues at statement 700.

```plaintext
PROGRAM MODEL
  SUBSCHEMA(AVERAGE)
  INVOKE
  PRIVACY(CFILE,MODE=I0,PRIVACY='XX99')
  OPEN(CFILE,ERR=100)
  IDENT='122-13-6704-01'
  START(CFILE,ERR=700)
  READ(CFILE,ERR=600,END=900)

700 PRINT *, 'ERROR ON START'

900 CLOSE(CFILE)
  TERMINATE
  END
```

Figure 3-12. Positioning a Realm

The first START statement omits the KEY option, which means CFILE will be positioned by primary key. The program sets the primary key (IDENT) to the value 122-13-6704-01 before the START statement is executed; the realm will be positioned to the record occurrence with the matching primary key value.

The second START statement includes the KEY option. The program sets the alternate key COURSE to the value PSY100. The first record occurrence in CFILE with an alternate key COURSE greater than or equal to PSY100 will be the one to which CFILE is positioned.

REWITING A RECORD

To rewrite a record, you must include a REWRITE statement in your program. The format is:

```
REWRITE(realm-name [,ERR=s])
```

where

- `s` = label of an executable statement to which control transfers on rewrite error

When a record is rewritten, the current values of those variables defined in the sub-schema are rewritten to the specified data base record. Data items defined in the schema but not defined in the sub-schema remain unchanged.

Before your program can rewrite a record, your program must have locked the record either with a record lock or with a realm lock. Typically, the record lock is used.

For a rewrite using a record lock, the program establishes the record locking mechanism by opening the realm for input/output. To rewrite the record, the program must include the following steps:

1. Read the record to the program's working storage area by executing a DML READ statement.
2. Set the value of each data item being changed to the appropriate new value.
3. Rewrite the record by executing a REWRITE statement.

When the realm is opened for input/output and the read is executed, CDCS expects an update operation and consequently locks the record. CDCS allows rewriting of only the locked record.

The program must not change the value of the primary key between the read and the rewrite of the record. The following example illustrates a processing sequence to avoid:

```
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT)
IDENT='200-44-7863-01'
REWRITE(CFILE)
```

Assuming a rewrite using a record lock, CDCS does not allow the rewrite in this example to be performed because the record is not locked; record 100-22-5860-04 is locked, but the rewrite is attempted on record 200-44-7863-01. CDCS issues an error diagnostic on the rewrite.

If an update requires that the value of a primary key be changed, the program must first delete the record with the old primary key value and then write the record with the new primary key value.

For a rewrite using a realm lock, the program establishes the realm lock by executing the LOCK statement. Then to rewrite a record, the program needs only to set the value of the primary key to the value of the record being rewritten and execute the REWRITE statement. The recommended rewriting procedure, however, includes more steps than these. The recommended procedure is the same as for a rewrite using a record lock: read the record, change the appropriate values, then rewrite the record. By reading the record, the program can test for an error on the read and, thereby, protect the integrity of the data base. With the realm lock, it is your responsibility to ensure that the program does not change the value of the primary key between the read and the rewrite of the record.
You should judiciously use a realm lock when rewriting records because the realm lock overrides the record lock and the checking capability available through the record lock.

In a program using the sample sub-schema AVERAGE (figure 3-1), the REWRITE statement appears as shown in figure 3-13. The program performs the rewrite by using a record lock. The program reads the record occurrence with a primary key value of 100-22-5860-04, changes the value of data item GRADE to the value 3.8, and then rewrites the record. In the rewritten record, all other values in the record occurrence remain unchanged. If an error occurs on rewrite, program execution continues at statement 800.

```
PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOKER
PRIVACY(CFILE,MODE=IO,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT,ERR=600)
GRADE=3.8
REWRITE(CFILE,ERR=800)
.
.
600 PRINT *, 'ERROR ON READ'
.
.
800 PRINT *, 'ERROR ON REWRITE'
.
.
900 CLOSE(CFILE)
TERMINATE
END
```

Figure 3-13. Rewriting a Record

DELETING A RECORD

To delete a record, you must include a DELETE statement in your program. The format is:

```
DELETE(realm-name C,ERR=s)
```

where

s = label of an executable statement to which control transfers on delete error

Before your program can delete a record, your program must have locked the record either with a record lock or with a realm lock. Typically, the record lock is used.

For a delete using a record lock, the program establishes the record locking mechanism by opening the realm for input/output. To delete the record, the program must include the following steps:

1. Read the record by executing a DML READ statement.
2. Delete the record by executing a DELETE statement.

When a realm is opened for input/output and the read is executed, CDCS expects an update operation and consequently locks the record. CDCS allows deletion of only the locked record.

The program must not change the value of the primary key between the read and the delete of the record. The following example illustrates a processing sequence to avoid:

```
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT)
IDENT='400-23-1248-07'
DELETE(CFILE)
```

Assuming a delete using a record lock, CDCS does not allow the delete in this example to be performed because the record is not locked; record 100-22-5860-04 is locked, but the delete is attempted on record 400-23-1248-07. CDCS issues an error diagnostic on the delete.

For a delete using a realm lock, the program establishes the realm locking mechanism by executing the LOCK statement. Then to delete a record, the program needs only to set the value of the primary key to the value of the record being deleted and execute the DELETE statement. The recommended procedure for deleting a record, however, includes more steps than these. The recommended procedure is the same as for a delete using a record lock: read the record and then delete the record. By reading the record, the program can test for an error on the read and, thereby, protect the integrity of the data base. With the realm lock, it is your responsibility to ensure that the program does not change the value of the primary key between the read and the delete of the record.

You should judiciously use a realm lock when deleting records because the realm lock overrides the record lock and the checking capability available through the record lock.

In a program using the sample sub-schema AVERAGE (figure 3-1), the DELETE statement appears as shown in figure 3-14. The program performs the delete by using a record lock. The program reads from CFILE the record occurrence with a primary key (IDENT) equal to 100-22-5860-04 and then deletes that record. If an error occurs on delete, program execution continues at statement 850.

USING DML TO PROCESS RELATIONS

Relation processing greatly simplifies programming when several related realms are required by the application program. Realms that have common data items can be joined in a relation. When a relation is included in a sub-schema, the relation can be accessed and read through DML. This means that a single relation read request by an application program returns a relation occurrence, which consists of one qualifying record from each of the realms comprising the relation.
PROGRAM MODEL
SUBSCHEMA(AVERAGE)
INVOLVE
PRIVACY(CFILE,MODE=I0,PRIVACY='XX99')
OPEN(CFILE,ERR=100)
IDENT='100-22-5860-04'
READ(CFILE,KEY=IDENT,ERR=600)
DELETE(CFILE,ERR=850)

600 PRINT *, 'ERROR ON READ'
850 PRINT *, 'ERROR ON DELETE'
900 CLOSE(CFILE)
TERMINATE
END

Figure 3-14. Deleting a Record

DML statements are available to provide for processing relations. The functions involved in relation processing are opening and closing the realms of the relation, positioning a relation through a start operation, and reading the relation. Paragraphs that follow describe these functions and the DML statements that you must include in your program to provide the functions. First, however, it is necessary to examine the structure of a relation and the manner in which CDMS returns records to the program's working storage area.

STRUCTURE OF A RELATION

A relation can be described as a hierarchical tree structure. The root of the tree is the realm through which the relation is entered; this is the first realm listed for a relation in the sub-schema. A data item in the realm at the root of the structure joins the realm to a common data item in the next realm listed for the relation. When the relation is entered, the value of the data item in the root realm record leads to a record in the second realm. More than one record in the second realm can contain the same value; thus one record in the root realm can lead to several records in the second realm.

The second realm in the relation can be joined to a third realm through a common data item. Once again, a record in the second realm can lead to several records in the third realm. This branching out from the root of the tree continues through each realm in the relation.

The tree structure of the three-realm relation REL3 of the university data base is illustrated in figure 3-15. The tree structure is normally pictured upside down, with the root at the top and branches going down. The first realm, PFILE, which is the root of the structure, consists of a master record for each professor. The second realm, CRSFILE, consists of a master record for each course. The third realm, CFILE, consists of curriculum records. The common data item joining the first and second realms is the professor identification; the common data item joining the second and third realms is the course identification.

Realms in a relation are numbered consecutively as ranks. The first realm entered (called the root realm) is always assigned rank 1. The rank is incremented by 1 for each successive realm in the relation. The value of the rank of a realm contrasts with the placement of the realm in the tree structure. The lower the rank, the higher the realm is shown in the tree structure; i.e., rank 1 (the lowest rank) is shown at the top of the tree structure. Figure 3-15 also shows ranks in the relation REL3.

When a relation is read, a record occurrence from each realm in the relation is returned to the program. A relationship exits between record occurrences in a relation: a parent/child relationship. A record occurrence that has another record occurrence at the next numerically higher rank in the relation is referred to as the parent record.
A record occurrence that has another record occurrence at the next numerically lower rank in the relation is referred to as the child record occurrence. In a parent/child relationship in relation REL3, a record occurrence in PFFile would represent the parent with corresponding record occurrences of CRSFile representing the children. Additionally, a record occurrence in Crsfile would represent the parent with corresponding record occurrences in CFile representing the children.

**Using the Sub-Schema**

The structure of a relation is defined when the schema is created. A relation that is available to an application program is included in the sub-schema. The sub-schema listing provides the names of the realms in the relation. A new sample sub-schema named COMPARISON, which makes available a three-realm relation, is shown in figure 3-16.

The sub-schema listing provides the following information:

- A relation is available to DML because the RELATION statement is included in the sub-schema.

- Three schema areas are joined by relation named REL3. The areas are named PROFESSOR, COURSE, and CURRICULUM in the schema; they are renamed as realms PFFile, CRsFile, and CFILe in the sub-schema. The order in which the areas (realms) appear in the Relation Statistics portion of the listing indicates the ranks of the realms: the first realm listed has the rank 1; the second, rank 2; and so forth.

- PFFile has a primary key named PROFID; CRsFile has an alternate key named PROF. Looking at the listing of aliases, you can see these fields both appear as PROF-ID in the schema. Obviously these fields represent unique professor identification and are common to both realms. You can assume that these items join the realms. The data administrator, however, should provide the common data items if they are not obvious and if programming considerations require that you know them.

- CRsFile has a primary key named CRSID; CFILe has an alternate key named COURSE. Looking at the listing of aliases, you can see these fields both appear as COURSE-ID in the schema. Obviously these fields represent unique course information and are common to both realms. You can assume that these items join the realms. The data administrator, however, should provide the joining data items under the conditions indicated previously.

- A restriction is placed on CRECORD. A relation occurrence will not be returned unless data item CODE contains the character C. Before a relation occurrence is returned to the program's working storage area, CDCS checks for restrictions and enforces any restrictions. CDCS allows only qualifying records to be returned.

**Opening a Relation**

Before your program can access any data records in an existing relation, the program must open the appropriate realms. To open all the realms in a relation, you can include a relation OPEN statement in your program. The format is:

```
OPEN(relation-name [,MODE=mode] [,ERR=s])
```

where

- **mode** = I (open for input only)
- **s** = label of an executable statement to which control transfers on open error

Your program should normally open a relation for input (MODE=I). The program should open the relation for input/output (MODE=IO) under two circumstances:

- Processing requirements indicate that the program should lock the records to prevent update during the relation read.
- The program updates individual realms in the relation following the relation read.

If your program is opening a relation in which one or more realms have controlled access, you must include in the program a PRIVACY statement for each realm that has controlled access.

The following statement opens for input the realms joined in relation REL3, which is shown in sample sub-schema COMPARISON (figure 3-16). If an error occurs on open, program execution continues at statement 50:

```
OPEN(REL3,MODE=I,ERR=50)
```

If you have included an OPEN statement in your program for each realm in the relation, you do not need to include a relation OPEN statement.

**Closing a Relation**

When your program has completed relation processing, the program must close the appropriate realms. To close all the realms of a relation, you can include a relation CLOSE statement in your program. The format is:

```
CLOSE(relation-name [,ERR=s])
```

where

- **s** = label of an executable statement to which control transfers on close error

The following statement closes the realms joined in relation REL3, which is shown in sample sub-schema COMPARISON (figure 3-16). If an error occurs on close, program execution continues at statement 60:

```
CLOSE(REL3,ERR=60)
```

If you include a CLOSE statement in your program for each realm in the relation, you do not need to include a relation CLOSE statement.
* SOURCE LISTING * (80351) DDLF 1.2+538.

```plaintext
00001 SUBSCHEMA COMPARE,SCHEMA=UNIVERSITY
00002 ALIAS-REALM PFILE=PROFESSOR
00004 ALIAS-RECORD PRECORD=PROF-REC
00005 ALIAS-ITEM PROFID=PROF-ID.PROF-REC
00006 ALIAS-ITEM PNAME=PROF-NAME
00008 ALIAS-REALM CRSFILE=COURSE
00010 ALIAS-RECORD CRSREC=COURSE-REC
00012 ALIAS-ITEM CRSID=COURSE-ID.COURSE-REC
00014 ALIAS-ITEM CRSNAME=COURSE-NAME
00016 ALIAS-ITEM PROF=PROF-ID.COURSE-REC
00018 ALIAS-ITEM FIELD=ACADEMIC-FIELD
00020 ALIAS-REALM CFILE=CURRICULUM
00022 ALIAS-RECORD CRECORD=CURR-REC
00024 ALIAS-ITEM CODE=COMPLETE-CODE
00026 REALM PFILE
00028 REALM CRSFILE
00030 REALM CFILE
00026 RECORD PRECORD
00027 CHARACTER*8 PROFID
00028 CHARACTER*30 PNAME
00029 CHARACTER*20 FIELD
00030 RECORD CRSREC
00031 CHARACTER*6 CRSID
00032 CHARACTER*20 CRSNAME
00033 CHARACTER*8 PROF
00034 RECORD CRECORD
00036 CHARACTER*14 IDENT
00037 CHARACTER*6 COURSE
00038 CHARACTER*1 CODE
00039 CHARACTER*8 DATE
00041 REAL GRADE
00042 RELATION REL3
00043 PROFID FOR AREA PFILE
00044 FIELD FOR AREA PFILE
00045 CRSID FOR AREA CRSFILE
00046 PROF FOR AREA CRSFILE
00047 IDENT FOR AREA CFILE
00048 COURSE FOR AREA CFILE
00049 GRADE FOR AREA CFILE
00043 RESTRICT CRECORD (CODE .EQ. 'C')
00044 END
```

Figure 3-16. Sub-Schema COMPARE (Sheet 1 of 2)
READING A RELATION

To read a relation, you must include a relation READ statement in your program. The format is:

```
READ(relation-name [,KEY symbol item-name] 
    [,ERR=s] [,END=s])
```

where

- **symbol** = = .EQ. .GT. .GE.
- **item-name** = primary or alternate key
- **s** = label of an executable statement to which control transfers on read error (ERR)
- **s** = label of an executable statement to which control transfers on end-of-file (END)

When you omit the KEY option, the read operation is sequential. When you include the KEY option, the read operation is random.

The END option is valid only for a sequential read operation.

Sequential Relation Read

A sequential relation read accesses the relation occurrence located at the current relation position. Successive read operations return relation occurrences by position of the root realm, which is the first realm listed for the relation in the Relation Statistics portion of the sub-schema listing. Indexed sequential files are sequenced in ascending primary key order, actual key files are sequenced by block and record slot within the block, and direct access files are sequenced by position in home blocks.

A relation occurrence is composed of record occurrences. A tree structure of record occurrences for relation REL3 is shown in figure 3-17. Assuming that A1 is the first record in PFILE, the first and subsequent sequential reads return record occurrences to the working storage area in the following order: A1B1C1, A1B1C2, A1B1C3, A1B2C4, and so forth. When record occurrences in CRSFILE and CFILE are exhausted, a subsequent sequential read returns the next record (A2, not shown) in PFILE and associated records in CRSFILE and CFILE as the operation repeats.

Typical FORTRAN 5 statements issued outside of a data base environment terminate with a fatal error if EOF is sensed and a test for EOF status is not included in the FORTRAN READ statement. If EOF status is not tested in a DML READ statement and EOF is sensed, program execution continues with the next statement. Consequently, it is necessary to test for EOF on a DML sequential read operation.

This can be handled in one of two ways:

- Include the END option on the READ statement. When an EOF is sensed, program execution continues at the statement specified in the option.
- Include a test for an EOF value of 100g in the data base status block. This option is described in section 4.

In a program using sample sub-schema COMPARE (figure 3-16), a sequential read appears as shown in figure 3-18. If an error occurs on read, program execution continues at statement 600. The END option is included on the READ statement to test for EOF. When EOF is reached, program execution continues at statement 900.

The sequential read returns the first record in PFILE and the first corresponding record occurrences in CRSFILE and in CFILE. Successive reads return qualifying record occurrences as indicated in the preceding discussion of the tree structure of record occurrences. If EOF is sensed on PFILE, the relation read transfers control to the statement specified by the END option.

Notice the PRIVACY statement. Since CFILE is joined in the relation and has controlled access, the privacy key for that realm is required.
Random Relation Read

A random relation read accesses a relation occurrence by the value of a referenced primary or alternate key. The referenced key must be in the root realm. For a program using the sample sub-schema COMPARE, the key named in the READ statement must be associated with PFILE (the root realm) rather than CRSFILE or CFILE. The program must set the value of the referenced key before the READ statement is executed.

In a program using sub-schema COMPARE (figure 3-16), a random read appears as shown in figure 3-19. The program sets the primary key PROFID of PFILE to RSS00860. The random read returns the record occurrence in PFILE that has PROFID equal to RSS00860 and the first corresponding record occurrences in CRSFILE and CFILE.

Control Break

A control break occurs when a new record occurrence is read for a parent realm in a relation. Control break status, however, is returned for the realm of the child. Therefore, if a realm in a relation has control break status after execution of a sequential read, the record occurrence read for this realm is a child record occurrence for a new parent record occurrence.

Null Occurrence

A null occurrence denotes that either no record occurrence qualifies for a read or that a record occurrence does not exist at a given level in a relation.

A read relation operation produces a null occurrence when one of the following is true:

- A parent record occurrence qualifies for the read, but no child record occurrence qualifies.
- A parent record occurrence qualifies for the read, but no child record occurrence exists.

Figure 3-19. Reading a Relation Randomly

In the example shown in the tree structure of record occurrences (figure 3-17), control break occurs when A1 is first read (when A1B1C1 is returned). In this situation, control break status is returned for CRSFILE (rank 2) and for CFILE (rank 3). A control break occurs when B2 is first read (when A1B2C4 is returned), when B3 is first read (when A1B3C6 is returned), and so forth. In these situations, control break status is returned for CFILE, which is rank 3 of the relation.

The presence of a control break and the rank of the realm that is the lowest ranked realm with control break status can be determined by checking the database status block. Status checking is described in section 4.
If a null record occurrence is returned for each realm in a relation except the root realm, another READ statement must be executed to obtain the next set of record occurrences.

A null occurrence consists of a display code right bracket (J) in each character position of the record in the working storage area. The presence of a null occurrence and the lowest rank on which it occurred can be detected by checking the data base status block. Status checking is described in section 4.

Some examples of null record occurrences returned are shown in figure 3-20. In the first example, the lowest rank with a null record occurrence is rank 2. In the second and third examples, the lowest rank with a null record occurrence is rank 3.

POSITIONING A RELATION

To position a relation for subsequent sequential read operations, you must include a START statement in your program. The format is:

```
START(relation-name,C,KEY symbol item-name)
```

where

- `symbol` = `.EQ.`, `.GT.`, `.GE.`
- `item-name` = primary or alternate key defined in the root realm
- `s` = label of an executable statement to which control transfers on start error

Before the START statement is executed, the realm must have been opened for input or input/output.

When you omit the KEY option, the relation is positioned by primary key value of the root realm; the root realm is positioned to the record occurrence with a primary key value equal to the current value of the primary key item. When you include the KEY option, the root realm is positioned to the first record occurrence with a matching key value.

In a program using the sample sub-schema COMPARE (figure 3-16), both forms of the START statement appear as shown in figure 3-21. If an error occurs on start, program execution continues at statement 700.

The first START statement omits the KEY option, which means the relation occurrence will be positioned by the root realm primary key. The primary key (PROFID) of the root realm is set to the value MLN00840 before the START statement is executed; the relation is positioned to the root realm record occurrence with the matching primary key.

The second START statement includes the KEY option. Alternate key FIELD is set to the value PSYCHOLOGY. The first record occurrence in PFILE with an alternate key equal to PSYCHOLOGY is the one to which root realm PFILE is positioned. The subsequent sequential reads reference the alternate key FIELD. These reads return record occurrences in the root realm in alphabetical order (collating sequence order) according to the value of the alternate key FIELD.

---

**Record Occurrences in a Relation**

```
Rank 1
   A1

Rank 2
   B1
      C1
      C2
      C3

A1 qualifies, B1 and B2 do not qualify.

A1 and B2 qualify.

A1 and B1 qualify, C1, C2, and C3 do not qualify.
```

```
Program's Working Storage Area
A1 B1
```

```
Program's Working Storage Area
A1 B2
```

```
Program's Working Storage Area
A1 33...3
```

```
Figure 3-20. Null Record Occurrence Examples
```
USING DML TO PROCESS DATA BASE TRANSACTIONS

Data base transactions can be used when you have many interrelated updates to perform on one or more data base files. These updates all need to be processed and made permanent to ensure the data is correct before other users access the updated records. A data base transaction is a convenient way for processing coordinated updates.

A group of updates for which the application program specifies the beginning and the completion is referred to as a data base transaction. At first, all updates within a data base transaction are considered temporary. These updates are considered permanent when the application program specifies the completion of the data base transaction (called committing a data base transaction). Figure 3-22 shows the sequence of operations in a data base transaction.

If an application program does not commit a data base transaction, but instead drops the data base transaction or terminates execution, each record that was updated within the data base transaction is restored to the state it was in just before the beginning of the data base transaction, and CDCS issues an informative diagnostic. There are several situations in which data base transactions are not committed. For example, program logic can determine that the data base transaction should not be committed and can cancel (drop) the data base transaction. System failure or program failure can occur during the application program's processing of the data base transaction. In each of these situations, updates made within the uncommitted data base transactions are reversed.

The application program can perform data base transactions only if transaction recovery files have been defined for the schema in the master directory. When a FORTRAN application program begins a data base transaction, CDCS processes subsequent update operations by that program in transaction mode. When processing in transaction mode, CDCS uses the exclusive record locking mechanism that prevents other users from accessing records updated within an uncommitted data base transaction.

PROCESSING OPERATIONS

FORTRAN DML statements provide for the operations involved in data base transactions. Three of these operations, which are directly involved in the program code dealing with updates, are described in the following paragraphs.

- Begin a data base transaction

Designates the beginning of a data base transaction and communicates a transaction identifier to CDCS. This causes CDCS to begin processing in transaction mode for the application program.

---

**UPDATING REALMS JOINED IN A RELATION**

Realms joined in a relation can be updated, but care should be exercised. Related files are joined in the schema by a common data item to form a parent/child relationship. The schema contains a JOIN clause in which a data item in one realm is equated with an identical data item in another realm. This common data item is called a join item.

CDCS normally does not monitor update operations that would alter the underlying relationship between related files. The exception is when constraints have been incorporated in the schema by the data administrator, as described in section 4.

Assuming constraints are not present, the following precautions should be noted:

- Modification of join item values can change parent/child relationships.
- Deletion of parent record occurrences can make all child record occurrences of the deleted parent record occurrence inaccessible when a relation is read.

Important rules to remember for relation update are:

- Always delete a child occurrence before deleting the parent record occurrence.
- Always write the parent record occurrence before writing a child record occurrence.
- Be aware of file positioning; input/output operations could alter positions on the files joined in the relation while within a sequential read relation loop.

---

**Figure 3-21. Positioning a Relation**

```
PROGRAM RELMOD
SUBSCHEMA(COMPARE)
INVOKE
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
PROFID='MLN00840'
START(REL3,ERR=700)
READ(REL3,ERR=600,END=900)
FIELD='PSYCHOLOGY'
START(REL3,KEY.EQ.FIELD,ERR=700)
READ(REL3,ERR=650,END=750)
CLOSE(REL3)
TERMINE
END
```

---

**Figure 3-22**
• Commit a data base transaction
  Designates the end of a data base transaction and indicates that the updates within the data base transaction are to be committed. This causes all the updates made within the data base transaction to be considered permanent.

• Drop a data base transaction
  Designates the end of a data base transaction and indicates that the updates already made within the data base transaction are to be cancelled. This causes each record updated within the data base transaction to be restored to the state it was in just before the beginning of the data base transaction.

**Beginning a Data Base Transaction**

To begin a data base transaction, you must use a `BEGINTRAN` statement. The format is:

```
BEGINTRAN (tran-id [,ERR=s])
```

where

- **tran-id** = character constant or variable
- **s** = label of an executable statement to which control transfers on begin error

A tran-id can be from 1 through 10 characters in length. If you use a character constant to specify a tran-id, enclose the character string in apostrophes. If you use a variable name to specify a tran-id, define the variable as type `CHARACTER*10`. Ensure that the tran-id is left-justified and blank filled in the field of the variable.

Records that are subsequently updated remain exclusively locked until the data base transaction is either completed or dropped. Updates are considered temporary until the data base transaction is successfully completed. If your program attempts to begin a data base transaction when the data administrator has not defined a transaction recovery file for the schema, a fatal error occurs.

**Committing a Data Base Transaction**

To commit a data base transaction, you must use a `COMMITTRAN` statement. The format is:

```
COMMITTRAN [(,ERR=s)]
```

where

- **s** = label of an executable statement to which control transfers on commit error

Execution of this statement causes all updates of the present data base transaction to become permanent; all record locks are released so that the records become available for access by other application programs (unless a realm lock applies).
In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-24. The program uses the BEGINTRAN statement to begin the data base transaction. If an error occurs in the program, execution continues at line 600 and the DROPTRAN statement is executed, causing the current data base transaction to be cancelled.

Dropping a Data Base Transaction

To cancel the current data base transaction, you must use a DROPTRAN statement. The format is:

DROPTRAN [(,ERR=s)]

where

s = label of an executable statement to which control transfers on cancel error

Execution of the DROPTRAN statement causes CDCS to restore the records updated within the data base transaction to their original states that existed just before the data base transaction was initiated and also causes CDCS to release all record locks.

Dropping a Data Base Transaction

To cancel the current data base transaction, you must use a DROPTRAN statement. The format is:

DROPTRAN [(,ERR=s)]

where

s = label of an executable statement to which control transfers on cancel error

Execution of the DROPTRAN statement causes CDCS to restore the records updated within the data base transaction to their original states that existed just before the data base transaction was initiated and also causes CDCS to release all record locks.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and DROPTRAN statements appear as shown in figure 3-24. The program uses the BEGINTRAN statement to begin the data base transaction. If an error occurs in the program, execution continues at line 600 and the DROPTRAN statement is executed, causing the current data base transaction to be cancelled.

Dropping a Data Base Transaction

To cancel the current data base transaction, you must use a DROPTRAN statement. The format is:

DROPTRAN [(,ERR=s)]

where

s = label of an executable statement to which control transfers on cancel error

Execution of the DROPTRAN statement causes CDCS to restore the records updated within the data base transaction to their original states that existed just before the data base transaction was initiated and also causes CDCS to release all record locks.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.

In a program using sample sub-schema COMPARE (figure 3-16), the BEGINTRAN and COMMITTRAN statements appear as shown in figure 3-23. The program uses the BEGINTRAN statement to begin the data base transaction. The program sets the primary key PROFID of PFILE to WLSN0855. PFILE is read randomly and the occurrence of this record is deleted. A new record is then written to PFILE. The CRSFILE is read randomly for any occurrence of the deleted PROF value. Each record with PROF=WLSN0855 is updated and rewritten with the PROF value of the new record. After all updates have been performed, the COMMITTRAN statement is executed and all updates become permanent.
USING DML FOR PROGRAM RESTART

FORTRAN DML statements provide for the operations involved in restarting programs. The restart component of data base transactions allows an application program to determine the point at which to begin a data base transaction following a system failure. An application program can determine whether or not a data base transaction was committed before the system failed. With this information available, the program can determine the point at which to resume a data base transaction.

The application program can perform a restart operation only if the data administrator has defined both a transaction recovery file and a restart identifier file in the schema.

PROCESSING OPERATIONS

The following paragraphs describe the operations used for program restart if a system failure occurs:

1. Obtain a restart identifier

Communicates with CDCS to obtain a restart identifier for the application program. The application program must save the restart identifier for subsequent use in a restart operation. If program restart capability is desired, this operation must be performed before the first data base transaction is begun.

2. Inquire about the status of the last data base transaction

Communicates to CDCS a restart identifier and obtains from CDCS the transaction identifier for the last completed data base transaction associated with that restart identifier. CDCS then assigns the restart identifier obtained to the program. This operation provides for restarting an application program. Application program logic then uses the transaction identifier to determine with which data base transaction to resume processing. The application program should contain the logic necessary to be restartable.

Assigning a Restart Identifier

To obtain the restart identifier assigned to this program by CDCS, you must use the ASSIGNID statement. The format is:

ASSIGNID (restart-id [,ERR=s3])

where

restart-id = variable

s = label of an executable statement to which control transfers on assign error

A restart-id can be from 1 through 10 characters in length. A variable name must be used to specify a restart-id and must be defined as type CHARACTER*10.

The restart identifier obtained by the ASSIGNID statement can then be used by the FINDTRAN statement; the program can then determine the status of a data base transaction when a system failure occurs. The restart identifier should not be saved on a data base file because it could be lost if failure occurs. The application program should contain the logic necessary to save the identifier outside of the program.

ASSIGNID should be specified before any updates are attempted within a data base transaction. ASSIGNID must not be specified within a data base transaction.

Performing a Restart Operation

To obtain information for a program restart operation after system failure, you must use the FINDTRAN statement. This statement is normally issued in the restart unit of the program. The format is:

FINDTRAN (restart-id, tran-id [,ERR=s3])

where

restart-id = character constant or variable

tran-id = variable

s = label of an executable statement to which control transfers on find error

Restart-id identifies the 1- through 10-character restart identifier that was assigned to the program by the ASSIGNID statement before the system failure. If you use a character constant to specify restart-id, enclose the character string in apostrophes. If you use a variable name to specify restart-id, define the variable as type CHARACTER*10.

Tran-id receives the transaction identifier of the last completed data base transaction; this identifier is returned only if the application program had begun a CDCS data base transaction prior to a system failure. The transaction identifier is not returned if no data base transaction has been committed for the specified restart identifier. A variable name must be used to specify tran-id and must be defined as type CHARACTER*10.

Tran-id receives the characters ********** (10 asterisks) if the restart identifier is unknown to CDCS. The restart identifier is unknown to CDCS if the wrong value is specified for restart-id or if the program terminated normally. If the program terminated normally, a new restart identifier must be obtained.

Tran-id receives a value of 10 blanks if the restart identifier is known to CDCS but no data base transaction had been completed prior to the system failure. The FINDTRAN statement executes normally, and a new restart identifier does not need to be obtained. The restart identifier specified as restart-id is reassigned to the program.

---

3-18

60483500 B
If a transaction identifier is returned, a new restart identifier does not need to be obtained. The restart identifier specified as restart-id is reassigned to the program.

In a program using sample sub-schema COMPARE (figure 3-16), the ASSIGNID and FINDTRAN statements appear as shown in figure 3-25. This program is run interactively, and the user must enter one of the three options given. If the initial option is chosen, the ASSIGNID statement obtains the restart identifier assigned by CDCS. Include in the job stream the control statements needed to create a file to save the restart identifier in case a system failure occurs. If the restart option is chosen, the FINDTRAN statement is executed, and the program can determine the status of the data base transaction when the system failure occurred. The file containing the restart identifier must be attached in the job stream when the restart option is chosen. If the end option is chosen, program execution is terminated.

```
PROGRAM RELMOD
CHARACTER RESTID *10
CHARACTER TRANID *10
CHARACTER OPTION *7
SUBSCHEMA (COMPARE)
INVOK
OPEN(2,FILE='RESTART')
10 PRINT*, 'ENTER: INITIAL, RESTART, OR END'
READ*, OPTION
IF (OPTION .EQ. 'INITIAL') THEN
   ASSIGNID (RESTID,ERR=50)
   WRITE(2,'(A10)') RESTID
   CLOSE(2)
   GO TO 25
ELSE IF (OPTION .EQ. 'END') THEN
   GO TO 60
ELSE IF (OPTION .EQ. 'RESTART') THEN
   READ(2,'(A10)') RESTID
   FINDTRAN (RESTID,TRANID,ERR=50)
   IF (TRANID .EQ. '**********') THEN
      PRINT*, 'RESTART UNSUCCESSFUL OR UNNECESSARY'
      GO TO 60
   ELSE
      ...
      ...
   END IF
   GO TO 10
ELSE IF (OPTION .EQ. 'END') THEN
   GO TO 60
ELSE IF (OPTION .EQ. 'RESTART') THEN
   READ(2,'(A10)') RESTID
   FINDTRAN (RESTID,TRANID,ERR=50)
   IF (TRANID .EQ. '**********') THEN
      PRINT*, 'RESTART UNSUCCESSFUL OR UNNECESSARY'
      GO TO 60
   ELSE
      ...
      ...
   END IF
   GO TO 10
ELSE
   ...
   ...
60 TERMINATE
END
```

Figure 3-25. Restarting a Data Base Transaction
DMS-170 offers a variety of error and status processing mechanisms. Each serves a specific purpose in the operating environment. These mechanisms are summarized in table 4-1 and detailed in the following paragraphs.

### USING ERR AND END PROCESSING OPTIONS

A transfer of control to special processing for error or end-of-file (EOF) conditions can be specified in your program. This is accomplished by including the ERR and END options in the appropriate DML statements.

The ERR option can appear in most statements, as shown in the formats in section 3. The END option can appear in the sequential READ statement.

The formats for the ERR and END options are:

- **ERR=**statement-label
- **END=**statement-label

When the ERR or END option is executed, control transfers to the statement identified by statement-label. The identified statement must be executable.

Assume an input/output error occurred during execution of the following statement:

```
OPEN(FILEX,ERR=50)
```

Execution of the OPEN statement is terminated, status is set to the appropriate error code as described later in this section, and program execution continues at statement 50.

### TABLE 4-1. ERROR AND STATUS PROCESSING MECHANISMS

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Definition</th>
<th>Programmer Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error processing option</td>
<td>Syntax option that passes control to program logic on an error condition. Control is not passed when a CDCS relation condition indicating a null record occurrence or control break occurs.</td>
<td>Include ERR option in appropriate DML statements.</td>
</tr>
<tr>
<td>End-of-file processing option</td>
<td>Syntax option that passes control to program logic on an EOF condition.</td>
<td>Include END option in appropriate DML statements.</td>
</tr>
<tr>
<td>Status block</td>
<td>An array to which CDCS returns data base status information.</td>
<td>Include the following operations in the program: establishing the data base status block, calling subroutine DMLDBST once, and testing the status block contents at appropriate points. Be aware of constraints, and follow the rules for modifying the files on which constraints have been imposed. If the program must have simultaneous locks on several resources, include a test for deadlock status and provide program logic to reestablish any released locks. Include the ASSIGNID statement and the FINDTRAN statement in the application program to determine the status of a data base transaction after a system or program failure occurs. Include the DROPTTRAN statement in the application program.</td>
</tr>
<tr>
<td>Constraint handling</td>
<td>A method of avoiding situations in which constraints could be violated.</td>
<td></td>
</tr>
<tr>
<td>Deadlock processing</td>
<td>A method of recovering from a situation in which programs are contending for locked resources.</td>
<td></td>
</tr>
<tr>
<td>Restart processing</td>
<td>A method of allowing an application program to determine the point at which to begin processing following a system or program failure.</td>
<td></td>
</tr>
<tr>
<td>Dropping a data base transaction</td>
<td>The capability to cancel a data base transaction.</td>
<td></td>
</tr>
</tbody>
</table>
Assume an EOF was sensed during execution of the following statement:

```
READ(FILEX,END=75)
```

Execution of the READ statement is terminated and execution continues at statement 75.

Several examples of this type of error and end-of-file processing appear throughout section 3.

**ESTABLISHING A DATA BASE STATUS BLOCK**

An array called a data base status block can be established in your program to receive data base status information. When the status block is included in your program, CDCS updates the block after every operation on a realm or a relation.

The minimum length of the data base status block is one word; the maximum length is 11 words. You can include some or all of the words for testing purposes. The content of the status block is shown in Table 4-2.

The following rules apply to the status block:

- Only one status block can exist at a time in the program.
- The status block must be declared as type INTEGER.
- The length of the block must be sufficient to completely include each desired portion of status information.

**TABLE 4-2. STATUS BLOCK CONTENT**

<table>
<thead>
<tr>
<th>Word</th>
<th>Content</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The CDCS or CRM octal error code for the last data base operation on a realm or a relation.</td>
<td>Only error codes are returned. Status codes indicating null occurrences or control breaks are not returned in this word. A zero value indicates no error occurred. Use 0 format for printing.</td>
</tr>
<tr>
<td>2†</td>
<td>A sub-schema item ordinal for CDCS errors occurring at the item level.</td>
<td>Item-level errors are associated with data validation and data base procedures established by the data administrator in the schema, and with CDCS record mapping. A zero value indicates no error occurred. Use I format for printing.</td>
</tr>
<tr>
<td>3†</td>
<td>A CRM octal code indicating file position of the realm when the last data base operation was performed. The code is returned for open, close, read, and start operations. For a relation operation, the code indicates the file position of the root realm.</td>
<td>Code values are: 018 Beginning-of-information. 10s End-of-keylist. The last primary key value associated with a given alternate key was returned during a read operation using an alternate key value. 208 End-of-record. A record was returned during a read operation. 100g End-of-information. A sequential read operation was attempted after the previous operation returned the last record in the realm. Use 0 format for printing.</td>
</tr>
<tr>
<td>4†</td>
<td>The severity of an error that occurred during the last data base operation.</td>
<td>A zero value indicates no error occurred or a non-fatal error occurred. A value of one indicates a fatal error occurred. Use 0 or I format for printing.</td>
</tr>
<tr>
<td>5</td>
<td>The name of the function being performed when an error or relation condition occurred; the name is left-justified and blank filled.</td>
<td>If no error has occurred, this word contains no valid information. Use A10 format for printing.</td>
</tr>
</tbody>
</table>
### TABLE 4-2. STATUS BLOCK CONTENT (Contd)

<table>
<thead>
<tr>
<th>Word</th>
<th>Content</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>6††</td>
<td>The rank of the realm on which a CDCS or CRM error occurred during a relation operation. (The ranks of realms joined in a relation are numbered consecutively, with the root realm having rank 1.)</td>
<td>A zero value indicates no error occurred. Use I format for printing.</td>
</tr>
<tr>
<td>7††</td>
<td>The lowest rank on which a control break occurred during a relation operation.</td>
<td>All realms in the relation with a rank greater than the rank stored in this word also have control breaks or null status. (Null status overrides control break status.) A zero value indicates no control break occurred. Use I format for printing.</td>
</tr>
<tr>
<td>8††</td>
<td>The lowest rank for which there was a null record occurrence during a relation operation.</td>
<td>All realms in the relation with a rank greater than the rank stored in this word also have null occurrences. A zero value indicates no null occurrence. Use I format for printing.</td>
</tr>
<tr>
<td>9,10,11</td>
<td>The name of the realm on which an error occurred; the name is left-justified and blank filled.</td>
<td>A blank value indicates no error occurred; a blank value also can indicate the error occurred on an operation not associated with input/output or occurred on an input/output operation not explicitly requested by the application program. Use A30 format for printing.</td>
</tr>
</tbody>
</table>

†Words 2, 3, and 4 are treated as a single unit by CDCS; length must be provided for all three words if information for any portion of the unit is to be returned.

††Words 6, 7, and 8 must all be defined to obtain any one word of relation status information.

The following declaration would provide a complete status block:

```
INTEGER STATUS(11)
```

The following declaration would provide a 5-word status block reflecting all information except that pertaining to relation processing:

```
INTEGER STAT(5)
```

The location and length of the status block are conveyed to CDCS through a call to the DMLDBST routine. The routine can be called at any point in the program after the INVOKE statement. The format of the call is:

```
CALL DMLDBST(block-name,length)
```

where

- `block-name` = name of the status block
- `length` = length in words of the status block

The following rules apply to the DMLDBST routine:

- The routine needs to be called one time only.
- The call to DMLDBST should appear before the first DML statement after INVOKE. Positioning of the DMLDBST call is important because the call initializes the status block to zeros and blanks.

- If DMLDBST is called more than once in a program, the status block defined in the last call is the one that is updated by CDCS.

In a program using the sample sub-schema COMPARE shown in section 3, a data base status block declaration appears as shown in figure 4-1. The formats for printing the contents of the data base status block are also shown in the figure.

```
PROGRAM DBSEXMP
INTEGER STATBLK(11)
SUBSCHEMA(COMPARE)
INVOKE
CALL DMLDBST(STABLK,11)
PRIVACY(CFILE,MODE=I,PRIVACY='XX99')
OPEN(REL3,MODE=I,ERR=100)
PROFID='MLN00840'
READ(REL3,ERR=600,END=900)
   ...
600 PRINT *, 'ERROR ON READ'
PRINT 700, STATBLK
700 FORMAT (1X, 'STATUS BLOCK' /
      1X, I5, 2X, I5, 2X, I5, 2X, A30)
900 CLOSE(REL3)
TERMINATE
END
```

Figure 4-1. Establishing a Data Base Status Block
ERROR CHECKING

Error checking should be performed after every operation on a realm or relation. Two methods are available:

- Test the error code in word 1 of the data base status block after every operation. For example:
  
  \[
  \text{OPEN(CFILE)} \\
  \text{IF(STATBLK(1) .NE. 0) ...}
  \]

- Include the ERR option on the DML statement as appropriate and handle status block printing in one specific section of the program. For example:
  
  \[
  \text{OPEN(CFILE,ERR=50) } \\
  \text{70 PRINT 60,STATBLK}
  \]

STATUS CHECKING

Status checking should be performed as appropriate during relation processing to determine control breaks and null occurrences. Testing is performed on words 7 and 8, respectively, of the data base status block. (For more information about control break and null occurrence, see section 3.)

Word 7 indicates the lowest rank on which a control break occurred. A nonzero value in this word indicates a control break. To test for a control break, you can include a test on word 7 in your program. For example:

\[
\text{READ(CFILE) } \\
\text{IF(STATBLK(7) .NE. 0) ...}
\]

Word 8 indicates the lowest rank for which there was a null occurrence. A nonzero value in this word indicates a null occurrence. Since the right bracket character (}) is stored in a null record, you would probably want your program to bypass printing or move spaces to the print line. To test for a null occurrence, you can include a test on word 8 in your program. For example:

\[
\text{READ(CFILE) } \\
\text{IF(STATBLK(8) .NE. 0) ...}
\]

AVOIDING CONSTRAINT VIOLATIONS

The data administrator incorporates constraints in the schema for the purpose of protecting interdependent data. Constraints can be defined for two logically associated items within a single file (single-file constraint) and for two logically associated items within two files (two-file constraint).

Consider an employment file in which each record occurrence contains an employee number and a manager number, where the manager number conforms to the structure of the employee number. Figure 4-2 illustrates this concept. Assume, for example, the data administrator designed the schema with the following single-file constraint:

\[
\text{MNGR-NO DEPENDS ON EMP-NO}
\]

In this example, MNGR-NO (the dependent item) is dependent upon EMP-NO (the dominant item). This means that no occurrence of the dependent record can exist in the data base unless an occurrence of the dominant record also exists with the same value of the associated data item. Also, no dominant record can be deleted if a dependent record exists with the same value of the associated data item.

The dominant item in a single-file constraint is always a primary key or an alternate key with no duplicates; the dependent item is a primary key or an alternate key, and the alternate key can have duplicates. You would violate the constraint presented in the example if you attempted to do any of the following:

- Store an employee EMPLOYMENT record if an EMPLOYMENT record for the referenced manager does not exist. (An organization could not recognize a manager who was not first an employee.)
- Change the value of the dominant item (EMP-NO) if a corresponding dependent item (MNGR-NO) exists. (An organization could not change an employee number as long as references to the old number existed.)
- Delete a manager EMPLOYMENT record if an employee EMPLOYMENT record with the corresponding manager number exists. (An organization could not remove a manager while an employee was still reporting to that individual.)

![Figure 4-2. Single-File Constraint Example](image-url)
In a single-file constraint, at least one record exists that has no dominant record. This situation occurs in the single-file constraint example for the employee who has no manager. The record for this employee must have the same value for both EMP-NO and MNGR-NO.

If you are creating a file on which a single-file constraint has been imposed, take the following steps in the order given:

1. Create the file with record occurrences of the items that have no dominant record.
2. Close the file.
3. Reopen the file for input/output and add the record occurrences of the dependent items. (Ensure that a dominant record occurrence exists before adding any corresponding dependent item.)

For a situation involving a two-file constraint, consider a course file and a curriculum file. Assume that the data administrator designed the schema with the following two-file constraint:

**COURSE-ID OF CURR-REC DEPENDS ON COURSE-ID OF COURSE-REC**

The records of the two files and the data items associated in the constraint are shown in figure 4-3. CURR-REC (the dependent record) is dependent upon COURSE-REC (the dominant record) if there is a correspondence between them. A correspondence exists if the dependent record and the dominant record each contain the same value for the common item, which is COURSE-ID in this example.

You would violate the constraint presented in the two-file constraint example if you attempted to do any of the following:

- Delete a COURSE-REC occurrence if a corresponding CURR-REC occurrence exists. (The university could not drop a course from its curriculum while a student was still enrolled.)
- Change the COURSE-ID value of a COURSE-REC occurrence if a corresponding CURR-REC occurrence exists. (The university could not change the identification code of a course as long as a student’s record still uses that code.)
- Add a CURR-REC occurrence if a corresponding COURSE-REC occurrence does not exist. (A student could not be enrolled in a course that was not being offered by the university.)

If you are modifying the common item of a file on which a two-file constraint has been imposed and the common item is a primary key, take the following steps in the order given:

1. Write the dominant record with the new value in the common item.
2. Read a dependent record, and change the value of the common item to the new value of the dominant record. Rewrite the dependent record. (Perform this step for each dependent record of the dominant record.)
3. Delete the dominant record with the old value.

If you are modifying the common item of a file on which a two-file constraint has been imposed and the common item is an alternate key, take the following steps in the order given:

1. Write each dependent record containing the old value of the item to a temporary file.
2. Delete each dependent record containing the old value of the item from the database.
3. Read the dominant record, and change the value of the data item to the new value. Rewrite the dominant record.
4. Read a dependent record from the temporary file, and change the value of the common item to the new value of the dominant record. Write the dependent record to the data base. (Perform this step for each dependent record of the dominant record.)

Since constraints are established in the schema and not indicated in any way in the sub-schema, it is the responsibility of the data administrator to supply you with this information. By being aware of constraints, you can anticipate violations and prevent them from occurring in your application program.

---

**Figure 4-3. Two-File Constraint Example**

| COURSE-REC (Course File) | \( \text{COURSE-ID (Primary Key)} \) | \( \text{COURSE-NAME} \) | \( \text{SCHOOL} \) | \( \ldots \) | \( \text{PROF-ID} \) |
|--------------------------|---------------------------------|----------------|----------------|----------|
| CURR-REC (Curriculum File) | \( \text{IDENT (Primary Key)} \) | \( \text{STUDENT-ID} \) | \( \text{COURSE-ID (Alternate Key)} \) | \( \ldots \) | \( \text{UNITS} \) |
When a constraint is violated, CDCS aborts the particular operation, returns a nonfatal 601g error code, and continues processing. The error message identifies the record on which the attempted violation occurred. Whenever you are writing, deleting, or rewriting a record, the appropriate data base status block entry should be tested.

Two general rules to remember for constraint processing are:

- Always delete a dependent record occurrence before deleting the dominant record occurrence.
- Always write the dominant record occurrence before writing a dependent record occurrence.

ANTICIPATING DEADLOCK SITUATIONS

CDCS allows concurrent access to a data base. This means that two or more application programs can access the same file (realm) at the same time. The following can take place:

- Two or more application programs can open the same file for input and perform simultaneous read operations.
- One application program can open a file for input/output and perform update operations, while other programs can open the same file for input and perform simultaneous read operations.
- Two or more application programs can open the same file for input/output, but only one program can gain immediate access to a particular record to perform update operations.

The integrity of the data base is maintained through CDCS locking mechanisms: the record locking mechanism and the file locking mechanism. CDCS holds a lock (either protected or exclusive) for an application program and prevents update of the locked file or record by any other program.

Exclusive locking prohibits read and update operations on the realm. Protected locking prohibits only update operations (allows read operations). See the CDCS 2 Application Programming Reference Manual for detailed information about locking.

Whenever two or more application programs contend for locked resources, which are files or records, a deadlock situation can occur. Contention occurs when two programs, each having at least one resource locked, attempt to lock a resource that is locked by the other program. Neither program can continue processing, because neither program can obtain the necessary locks. CDCS automatically releases the locked resources of one program. The other program then can obtain the locks it requires and can continue processing.

When CDCS has detected a deadlock situation and has released the locked resources of an application program, CDCS issues the deadlock error status code 6638 to that program. If the application program established the data base status block, the program can check the first word for the deadlock code.

If your program must have locks on several resources, your program should always test for deadlock status before attempting to update a file. If deadlock occurs, your program should reestablish the locks that it held before continuing further processing.

An example illustrating deadlock processing appears in figure 4-4. Files joined in relation REL3 are opened for input/output. The program presumably is reading a record prior to update and CDCS has locked all records in the relation occurrence. The example includes a test of word 1 in the status block to enter a loop in case of deadlock. In the loop, the program attempts to reestablish the locks and checks for deadlock.

```
PROGRAM DEADLOCK
INTEGER STATBLK(11)
SUBSCHEMA(COMPARE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(REL3,ERR=100)
PROFID='JMS00160'
30 READ(REL3,KEY=PROFID)
IF(STATBLK(1).EQ.0"663") GOTO 30
900 CLOSE(REL3)
TERMINATE
END
```

Figure 4-4. Deadlock Processing
<table>
<thead>
<tr>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>An application program opens a realm for input/output and includes a DML LOCK statement. (This should be avoided whenever possible.)</td>
<td>CDCS locks the entire realm against update by other users. An unlock or close operation by that program releases the lock.</td>
</tr>
<tr>
<td>An application program opens a realm for input/output without including a DML LOCK statement.</td>
<td>CDCS locks the record on the read operation. A rewrite, delete, or another read operation by that program releases the lock.</td>
</tr>
<tr>
<td>An application program opens a realm for output without including a DML LOCK statement.</td>
<td>CDCS locks the entire realm. A close operation by that program releases the lock.</td>
</tr>
<tr>
<td>An application program opens a relation for input/output.</td>
<td>CDCS locks all records in a given relation occurrence. A rewrite or delete operation by the program releases the lock on the record updated. The next relation read operation by that program releases the record locks on the files for which no new record has been read.</td>
</tr>
</tbody>
</table>

Whenever two or more application programs contend for locked resources, which are files or records, a deadlock situation can occur. Contention occurs when two programs, each having at least one resource locked, attempt to lock a resource that is locked by the other program. Neither program can continue processing, because neither program can obtain the necessary locks. CDCS automatically releases the locked resources of one program. The other program then can obtain the locks it requires and can continue processing.

When CDCS has detected a deadlock situation and has released the locked resources of an application program, CDCS issues the deadlock error status code 663g to that program. If the application program established the data base status block, the program can check the first word for the deadlock code.

If your program must have locks on several resources, your program should always test for deadlock status before attempting to update a file. If deadlock occurs, your program should reestablish the locks that it held before continuing further processing.

An example illustrating deadlock processing appears in figure 4-5. Files joined in relation REL3 are opened for input/output. The program presumably is reading a record prior to update and CDCS has locked all records in the relation occurrence. The example includes a test of word 1 in the status block to enter a loop in case of deadlock. In the loop, the program attempts to reestablish the locks and checks for deadlock.

```plaintext
PROGRAM DEADLCK
INTEGER STATBLK(H)
SUBSCHEMA(COMPARE)
INVOKE
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(REL3,ERR=100)
PROFID='JMS00160'
30 READ(REL3,KEY=PROFID)
IF(STATBLK(1) .EQ. '0''663'') GO TO 30
     .
     .
900 CLOSE(REL3)
TERMINATE
END
```

Figure 4-5. Deadlock Processing
DEVELOPING FORTRAN PROGRAMS

FORTRAN application programming in the DMS-170 environment relieves you of several responsibilities. For example:

- You do not have to describe data within your program; the data administrator incorporates data descriptions in the schema and sub-schema. Data descriptions in a sub-schema are included in your program.
- You do not have to write conversion routines; CDCS handles all conversion for you.
- You do not have to write all routines that perform validity checking; the data administrator generates data base procedures, which are specified in the schema and called at appropriate times.
- You do not have to write separate logging and recovery utilities; the data administrator provides for data base restoration by specifying logging operations in the master directory.
- You do not have to be concerned with the details of input/output; CDCS handles them.

DEVELOPING AN APPLICATION PROGRAM

To develop a DMS-170 application program, you must do the following:

- Obtain a listing of the sub-schema from the data administrator.
- Obtain the name of the sub-schema library from the data administrator.
- Obtain the appropriate privacy keys from the data administrator.
- Be aware of any constraints that have been incorporated in the schema.
- Include appropriate DML statements in your FORTRAN program.
- Obtain information on whether data base transactions can or should be used.

COMPILING AND EXECUTING THE SOURCE PROGRAM

To compile and execute a DMS-170 FORTRAN application program, you must do the following:

1. Attach the sub-schema library for DML preprocessing of the source program.
2. Include a DML control statement for DML preprocessing of the source program.
3. Include an FTNS control statement that specifies the DML output file as the input file for the FORTRAN 5 compiler.
4. Include an LDSET control statement for loading the system library for execution of the source program.
5. Include the name of the file containing the relocatable binary program (LGO is the default name) to execute the program.
6. Be sure that CDCS is active or use CDCSBTF under the direction of the data administrator.

DML statements are preprocessed before source program compilation. The DML preprocessor translates the DML statements into appropriate FORTRAN statements. When translation is complete, the DML preprocessor writes the FORTRAN source program to an output file with the default name DMLOUT. This output file, complete with translated DML statements, becomes the input file to the FORTRAN compiler. A block diagram illustrating FORTRAN/DML preprocessing is shown in figure 5-1.

The DML control statement calls the DML preprocessor. A list of DML control statement parameters is shown in figure 5-2.

The statements required to execute the DML preprocessor and to compile the source program are shown in figure 5-3.

The statements required to execute the DML preprocessor and to compile and execute the source program are shown in figure 5-4. An LDSET control statement naming the system library, DMSLIB, must be included for program execution.

![Figure 5-1. FORTRAN DML Preprocessing](image-url)
**Figure 5-2. DML Control Statement**

<table>
<thead>
<tr>
<th>p1</th>
<th>SB=lfn</th>
<th>Name of file containing sub-schema library.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SB</td>
<td>Same as SB=SBLFN.</td>
</tr>
<tr>
<td></td>
<td>SB=0</td>
<td>Not allowed.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Same as SB=SBLFN.</td>
</tr>
<tr>
<td>p2</td>
<td>LV=F5</td>
<td>Specifies FORTRAN 5.</td>
</tr>
<tr>
<td></td>
<td>LV</td>
<td>Same as LV=F5.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Dependent on installation.</td>
</tr>
<tr>
<td>p3</td>
<td>I=lfn</td>
<td>Name of file containing FORTRAN source program with added DML statements to be preprocessed by DML.</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>Same as I=COMPILE.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Same as I=INPUT.</td>
</tr>
<tr>
<td></td>
<td>I=0</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>p4</td>
<td>0=lfn</td>
<td>Name of file to which translated version of FORTRAN source program is to be written. DML statements appearing in FORTRAN program are translated into FORTRAN statements before being written to this file.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Same as 0=DMLOUT.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Same as 0=DMLOUT.</td>
</tr>
<tr>
<td></td>
<td>0=0</td>
<td>No output is produced.</td>
</tr>
<tr>
<td>p5</td>
<td>E=lfn</td>
<td>Name of file to which error diagnostics are to be written.</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>Same as E=ERRS.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Same as E=OUTPUT.</td>
</tr>
<tr>
<td>p6</td>
<td>ET=op</td>
<td>Error termination code. Four levels of errors are defined; if an error of the specified level or higher takes place, the job is aborted to an EXIT(S) control statement (NOS/BE) or EXIT control statement (NOS). The abort does not take place until DML is finished. The possible values for op, in increasing order of severity, are as follows:</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>Trivial. The syntax of the usage is correct, but it is questionable.</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>Warning. The syntax is incorrect, but the processor has been able to recover by making an assumption about what was intended.</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>Fatal. An error prevents DML from processing the statement in which it occurs. Unresolvable semantic errors also fall into this category. Processing continues with the next statement.</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>Catastrophic. Compilation cannot continue; however, DML advances to the end of the current program unit and attempts to process the next program unit.</td>
</tr>
<tr>
<td></td>
<td>ET</td>
<td>Same as ET=F.</td>
</tr>
<tr>
<td></td>
<td>ET=0</td>
<td>The job step is not to be aborted even if errors occur (except for control statement errors).</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Same as ET=0.</td>
</tr>
<tr>
<td>p7</td>
<td>DS</td>
<td>Directive suppression. Listing control directives are not generated; all FORTRAN statements generated by DML preprocessing appear in the FORTRAN source listing.</td>
</tr>
<tr>
<td></td>
<td>omitted</td>
<td>Listing control directives are generated; FORTRAN statements generated by DML preprocessing of the SUBSCHEMA and INVOKE statements do not appear in the FORTRAN source listing.</td>
</tr>
<tr>
<td></td>
<td>DS</td>
<td>FORTRAN CALL statements generated as a result of executable DML statements always appear on the FORTRAN source listing regardless of DS specification.</td>
</tr>
</tbody>
</table>
SAMPLE PROGRAMS

Sample programs appear in the remainder of this section. Each program uses the data base environment that is established and illustrated in appendix C. You should read this appendix to become familiar with the schema, sub-schemas, and stored data before examining the FORTRAN programs.

When the DML preprocessor translates DML statements into FORTRAN statements, the FORTRAN statements can be printed out or suppressed, depending on the setting of the DS parameter on the DML control statement. When the DS parameter is included, all FORTRAN statements generated by the DML preprocessor appear in the FORTRAN source listing. When the DS parameter is omitted, listing control statements are generated and inserted immediately after the SUBSCHEMA and INVOKE statements; therefore, the FORTRAN statements generated by DML preprocessing of these statements do not appear in the FORTRAN source listing.

Listing control directives appear in the sample program source listings in the following form:

- \texttt{CS \ LIST(ALL=0)}
- \texttt{CS \ LIST(ALL)}

These directives are generated automatically by the DML preprocessor. They appear because the DS parameter in each DML control statement was omitted. Notice, however, that CALL statements generated as a result of executable DML statements appear regardless of the DS parameter setting.

Each sample program is illustrated by including the control statements, the source program statements, the compilation listing, and the output of program execution. The programs are:

- Program RATING Figure 5-5
- Program INDAVGE Figure 5-6
- Program RELATE Figure 5-7
- Program CHARGES Figure 5-8
- Program ADMIT Figure 5-9
- Program TRANPRG Figure 5-10

Program TRANPRG, shown in Figure 5-10, is an interactive job. The file description and input file for this program appear as shown in Figure 5-11.

The figures show the sample programs (listed above) being executed when CDCS is active at system control point. CDCS Batch Test Facility (CDCSBTF) can also be used. When using CDCSBTF, replace the LDSET and the LGO control statements with the following control statements:

- \texttt{LIBRARY, DMSLIB.}
- \texttt{CDCSBTF(LGO/MPFN=MSTRDIR, \texttt{\texttt{UN=xx})}}
Control Statements

Job Statement
USER statement
CHARGE statement
ATTACH(SSI);LIB)
DML(SB=SSLIB,LV=F5)
FTNS(I=DMLOUT)
LDSET(LIB=DMLIB)
END.
End-of-record

Source Program

PROGRAM RATING

C
C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES THE
C AVERAGE FOR THE SCHOOL. THE PROGRAM PERFORMS A SEQUENTIAL
C READ OF ALL STUDENT GRADES. THE END PARAMETER ON THE DML.
C READ STATEMENT TRANSFERS CONTROL TO STATEMENT 70 ON EOF.
C THE PROGRAM THEN Calculates THE AVERAGE AND PRINTS OUT
C THE SOLUTION.
C
INTEGER STATBLK(11)
SUBSCHEMA(AVERAGE)
INVOKES(DMLDBST(STATBLK,11)
PRIVACY(CF,PRIVACY='XX99')
OPEN(CF,ERR=50)
N=0
TOTAL=0
READ(CF,ERR=50,END=70)
IF(GRADE.EQ.0.0) GO TO 20
N=N+1
TOTAL=TOTAL + GRADE
GO TO 20
50 PRINT 60,STATBLK
60 FORMAT (1X,'STATUS BLOCK' /
 1 1X,04,2X,15,2X,03,2X,12,2X,
 1 A10,2X,15,2X,15,2X,15,2X,A30)
GO TO 30
70 TOTAL=TOTAL / N
PRINT 75,TOTAL
75 FORMAT (1X, 'AVERAGE IS' /
 1 1X,F4.1)
80 CLOSE(CF)
TERMINATE
END
End-of-record

Figure 5-5. Program RATING (Sheet 1 of 3)
Compilation Listing

PROGRAM RATING 76/176 OPT=0 FTN 5.1+538

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

Figure 5-5. Program RATING (Sheet 2 of 3)
<table>
<thead>
<tr>
<th>COURSE</th>
<th>DBFO001</th>
<th>DBIO01</th>
<th>DBREAL</th>
<th>DBRELST</th>
<th>DBRLST</th>
<th>IdENT</th>
<th>N</th>
<th>STATBLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B</td>
<td>15B</td>
<td>0B</td>
<td>0B</td>
<td>10B</td>
<td>7B</td>
<td>0B</td>
<td>220B</td>
<td>205B</td>
</tr>
<tr>
<td>/DB0001/</td>
<td>/DB0000/</td>
<td>/DB0001/</td>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB0001/</td>
<td>/DB0001/</td>
<td>/DB0001/</td>
</tr>
<tr>
<td>CHAR*6</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>CHAR*14</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>DBRO001</td>
<td>DBSCNAM</td>
<td>DBSTAT</td>
<td>DBSO001</td>
<td>DBT0001</td>
<td>G R A D E</td>
<td>STUDENT</td>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>11B</td>
<td>4B</td>
<td>3B</td>
<td>14B</td>
<td>6B</td>
<td>0B</td>
<td>1B</td>
<td>221B</td>
<td></td>
</tr>
<tr>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB0000/</td>
<td>/DB001AA/</td>
<td>/DB0001/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBRO001</td>
<td>DBSCNAM</td>
<td>DBSTAT</td>
<td>DBSO001</td>
<td>DBT0001</td>
<td>GRADE</td>
<td>STUDENT</td>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>REAL</td>
<td>CHAR*11</td>
<td>REAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### PROCEDURES (LO=A)

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>ARGs</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMLCLS</td>
<td>2</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>DMLDBST</td>
<td>2</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>DMLEND</td>
<td>0</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>DMLINV</td>
<td>6</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>DMLOPN</td>
<td>4</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>DMLPRV</td>
<td>8</td>
<td>SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>LOCF</td>
<td>GENERIC</td>
<td>1</td>
<td>INTRINSIC</td>
</tr>
</tbody>
</table>

### STATEMENT LABELS (LO=A)

<table>
<thead>
<tr>
<th>LABEL</th>
<th>ADDRESS</th>
<th>PROPERTIES</th>
<th>DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>40B</td>
<td>73</td>
<td>70</td>
</tr>
<tr>
<td>50</td>
<td>56B</td>
<td>78</td>
<td>75</td>
</tr>
<tr>
<td>60</td>
<td>110B</td>
<td>FORMAT</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>88</td>
</tr>
</tbody>
</table>

### ENTRY POINTS (LO=A)

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>ARGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATING</td>
<td>5B 0</td>
<td></td>
</tr>
</tbody>
</table>

### STATISTICS

- PROGRAM-UNIT LENGTH = 2228 = 146
- SCM LABELLED COMMON LENGTH = 67B = 55
- SCM STORAGE USED = 607008 = 25024
- COMPILe TIME = 0.039 SECONDS

Output From Program Execution

AVERAGE IS

3.6

Figure 5-5. Program RATING (Sheet 3 of 3)
Control Statements for Interactive Job

ATTACH(FTNRUN)
ATTACH(SSLIB)
DML(SB=SSLIB, LV=F5, I=FTNRUN)
FTNSCI=DMLOUT)
ATTACH(INTRAN)
FILE(INTRAN, RT=Z, BT=C)
LDSET(LIB=DMSLIB)
LGO.
End-of-record

Source Program

PROGRAM TRANPRG
C
C THIS PROGRAM DEMONSTRATES THE USE OF TRANSACTIONS AND
C PROGRAM RESTART. FIRST THE PROGRAM DETERMINES IF THE
C RUN IS AN INITIAL RUN OR A RESTART OPERATION BY
C REQUESTING INPUT FROM A TERMINAL.
C
C THEN THE PROGRAM READS TRANSACTIONS FROM FILE INTRAN
C (SHOWN IN FIGURE 5-11), BEGINS TRANSACTION
C PROCESSING, AND UPDATES TWO REALMS: SFILE AND CFILE.
C
C DURING RESTART PROCESSING, THE PROGRAM POSITIONS FILE
C INTRAN BY READING AND DISCARDING RECORDS THAT WERE
C SUCCESSFULLY PROCESSED BEFORE THE FAILURE.
C
INTEGER STATBLK(11)
CHARACTER RESTID *10
CHARACTER TRANID *10
CHARACTER INID *10
CHARACTER INSTID *11
CHARACTER INMJR *20
CHARACTER OPTION *7
SUBSCHEMA (RELATION)
DATA TRANID/'0000000000'/
INVOC
CALL DMLDBST(STATBLK,11)
OPEN(5,FILE='INTRAN',STATUS='OLD', ACCESS='SEQUENTIAL')
OPEN(2,FILE='RESTART')
10 PRINT*, ' ENTER: INITIAL, RESTART, OR END'
READ*, OPTION
IF (OPTION .EQ. 'INITIAL') THEN
ASSIGNID (RESTID, ERR=50)
WRITE(2,'(A10)') RESTID
CLOSE(2)
GO TO 25
ELSE IF (OPTION ,EQ. 'RESTART') THEN
READ(2,'(A10)') RESTID
FINDTRAN (RESTID,TRANID, ERR=50)
IF (TRANID ,EQ. '***********') THEN
PRINT*, ' RESTART UNSUCCESSFUL OR UNNECESSARY'
GO TO 60
ELSE
READ (5,,ERR=60,END=60) INID,INSTID,INMJR,NUM
DO 20 II=1,NUM
READ (5,,ERR=60,END=60) IDENT,COURS,GRADE,
CODE,DATE,UNITS
IF (INID ,EQ. TRANID) THEN
GO TO 25
ELSE
GO TO 15
END IF
END IF

Figure 5-10. Program TRANPRG (Sheet 1 of 5)
ELSE IF (OPTION .EQ. 'END') THEN
  GO TO 60
ELSE
  GO TO 10
END IF

BEGIN DATA BASE PROCESSING

25 PRIVACY (CFILE,PRIVACY='XX99')
OPEN (SFILE,MODE=IO,ERR=50)
OPEN (CFILE,MODE=IO,ERR=50)

MAIN LOOP BEGINS. THIS READS AND PROCESSES FILE INTRAN.

DO 35 JJ=1,9999
READ (5,*,ERR=45,END=55) TRANID, INSTID,INMJR,NUM
BEGINTRAN (TRANID,ERR=45)
STID=INSTID
READ (SFILE,KEY=STID,ERR=45)
INMJR=MAJOR
REWRITE (SFILE,ERR=45)
CSTID=STID
DO 30, 11=1,NUM
READ (5,*,ERR=45,END=45) IDENT,COURS,GRADE,
  CODE,DATE,UNITS
30 WRITE (CFILE,ERR=45)
35 COMMITTRAN (ERR=45)

MAIN LOOP ENDS.

45 PRINT*, ' TRANSACTION ERROR, TRANID = ', TRANID
50 PRINT 98, STATBLK(1),STATBLK(2),STATBLK(3)
55 PRINT*, 'DATA BASE PROCESSING COMPLETED'
CLOSE (SFILE)
CLOSE (CFILE)
60 TERMINATE
CLOSE (5,STATUS='DELETE')
90 FORMAT (A10,A11,A20,I1)
92 FORMAT (A14,A6,F3.1,A1,A8,I1)
98 FORMAT (1X,'STATUS BLOCK'/1X,04,2X,I5,2X,A10)
END

Compilation Listing

PROGRAM TRANPRG 74/74 OPT=0

1  PROGRAM TRANPRG
2  C  THIS PROGRAM DEMONSTRATES THE USE OF TRANSACTIONS AND
3  C  PROGRAM RESTART. FIRST THE PROGRAM DETERMINES IF THE
4  C  RUN IS AN INITIAL RUN OR A RESTART OPERATION BY
5  C  REQUESTING INPUT FROM A TERMINAL.
6  C  THEN THE PROGRAM READS TRANSACTIONS FROM FILE INTRAN
7  C  (SHOWN IN THE PRECEDING FIGURE), BEGINS TRANSACTION
8  C  PROCESSING, AND UPDATES TWO REALMS: SFILE AND CFILE.
9  C  DURING RESTART PROCESSING, THE PROGRAM POSITIONS FILE
10  C  INTRAN BY READING AND DISCARDING RECORDS THAT WERE
11  C  SUCCESSFULLY PROCESSED BEFORE THE FAILURE.
12  C

Figure 5-10. Program TRANPRG (Sheet 2 of 5)
BEGIN DATA BASE PROCESSING
PRIVACY (CFILE,PRIVACY='XX99')
OPEN (SFILE,MODE=IO,ERR=50)
CALL DMLOPN(DBF0001,0001,2HI0,*50 )
OPEN (CFILE,MODE=IO,ERR=50)
MAIN LOOP BEGINS. THIS READS AND PROCESSES FILE INTRAN.
CALL DMLOPN(DBF0002,0002,2HI0,*50 )
DO 35 JJ=1,9999
READ (5,*,ERR=45,END=55) TRANID, INSTID,INMJR,NUM
BEGINTRAN (TRANID,ERR=45)
CALL DMLBEG(TRANID ,45 )
STID=INSTID
READ (SFILE,KEY=STID,ERR=45)
CALL DMLRDK(DBF0001,0001,0001,0001,1,0011,0,0000,00,
+STID ,45 )
```
183 ** INMJR=MAJOR
184 ** REWRITE (SFILE, ERR=45)
185 CALL DMLREW (DBF0001, 0, 00001, 000001, 45 )
186 CSTID=STID
187 DO 30, II=1, NUM
188 READ (5, *, ERR=45, END=45) IDENT, COURS, GRADE,
189 x CODE, DATE, UNITS
190 ** WRITE (CFILE, ERR=45)
191 30 CALL DMLWRT (DBF0002, 0, 0002, 00001, 45 )
192 ** COMMITTRAN (ERR=45)
193 C
194 C MAIN LOOP ENDS.
195 C
196 35 CALL DMLCMT (+45 )
197 45 PRINT*, ' TRANSACTION ERROR, TRANID = ', TRANID
198 50 PRINT 98, STATBLK (1), STATBLK (2), STATBLK (3)
199 ** DROTRAN
200 CALL DMLDRP
201 55 PRINT*, 'DATA BASE PROCESSING COMPLETED'
202 ** CLOSE (SFILE)
203 CALL DMLCLS (DBF0001, 0001)
204 ** CLOSE (CFILE)
205 CALL DMLCLS (DBF0002, 0002)
206 ** TERMINATE
207 60 CALL DMLEND
208 CLOSE (5, STATUS='DELETE')
209 90 FORMAT (A10, A11, A20, I1)
210 92 FORMAT (A14, A6, F3.1, A1, A8, I1)
211 98 FORMAT (1X, 'STATUS BLOCK'/1X, 04, 2X, I5, 2X, A10)
212 END

---VARIABLE MAP--- (LO=A)
-NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE

| CODE   | 0B /D0002AB/ | CHAR*1 |
| COURS  | 2B /D0002/ | CHAR*6 |
| CSTID  | 1B /D0002/ | CHAR*11 |
| DATE   | 0B /D0002AB/ | CHAR*8 |
| DBAG001 | 706B | INTEGER 3 |
| DBF0001 | 16B /D0000/ | INTEGER 35 |
| DBF0002 | 67B /D0000/ | INTEGER 35 |
| DBI0001 | 0B /D0001/ | CHAR*1 |
| DB10002 | 0B /D0002/ | CHAR*1 |
| DBN0001 | 134B /D0000/ | INTEGER 3 |
| DBREALM | 0B /D0000/ | INTEGER 3 |
| DBRELS | 1B /D0000/ | INTEGER 2 |
| DBRUID | 7B /D0000/ | INTEGER 3 |
| DBR0001 | 12B /D0000/ | INTEGER 3 |
| DBR0002 | 63B /D0000/ | INTEGER 3 |
| DBSCNAM | 4B /D0000/ | INTEGER 3 |
| DBSTAT | 3B /D0000/ | INTEGER |
| DBS0001 | 15B /D0000/ | INTEGER |
| DBS0002 | 66B /D0000/ | INTEGER |
| DBT0001 | 61B /D0000/ | INTEGER 2 |
| DBT0002 | 132B /D0000/ | INTEGER 2 |
| GRADE | 0B /D000000/ | REAL |
| IDENT | 0B /D000000/ | CHAR*14 |
| II | 712B | INTEGER |
| INID | 700B | CHAR*10 |
| INMJR | 703B | CHAR*20 |
| INSTID | 701B | CHAR*11 |
| JJ | 714B | INTEGER |
| MAJOR | 1B /D00001/ | CHAR*20 |
| NUM | 711B | INTEGER |
| OPTION | 703B | CHAR*7 |
| RESTID | 676B | CHAR*10 |
| STATBLK | 663B | INTEGER 11 |
| STD | 0B /D000000/ | CHAR*11 |
| TRANID | 677B | CHAR*10 |
| UNITS | 0B /D000000/ | INTEGER |
```

Figure 5-10. Program TRANPRG (Sheet 4 of 5)
PROCEDURES—(LO=A)

NAME TYPE ARGS CLASS
DMLBE 2 SUBROUTINE
DMLCLS 2 SUBROUTINE
DMLCMT 1 SUBROUTINE
DMLDBST 2 SUBROUTINE
DMLDRP 0 SUBROUTINE
DMLEND 0 SUBROUTINE
DMLFIND 3 SUBROUTINE
DMLGTID 2 SUBROUTINE
DMLINV 6 SUBROUTINE
DMLOPN 4 SUBROUTINE
DMLOPRV 8 SUBROUTINE
DMLRDK 11 SUBROUTINE
DMLREM 5 SUBROUTINE
DMLRWT 5 SUBROUTINE
LOCF GENERIC 1 INTRINSIC

STATEMENT LABELS—(LO=A)

LABEL ADDRESS PROPERTIES
DEF 10 46B 131
15 112B 147
20 INACTIVE DO-TERM 149
25 146B 166
30 INACTIVE DO-TERM 191
35 INACTIVE DO-TERM 196
40 255B 197
50 257B 198
55 263B 201
60 271B 207
90 345B FORMAT 209
92 350B FORMAT 210
98 354B FORMAT 211

ENTRY POINTS—(LO=A)

NAME ADDRESS ARGS
TRANPRG 58 0

I/O UNITS—(LO=A)

NAME PROPERTIES
TAPE2 AUX/FMT/SEQ
TAPE5 AUX/FMT/SEQ

STATISTICS—

PROGRAM-UNIT LENGTH 717B = 463
CM LABELLED COMMON LENGTH 152B = 106
CM STORAGE USED 63600B = 26496
COMPILE TIME 0.185 SECONDS

Output and Interactive Response From Program Execution

ENTER: INITIAL, RESTART, OR END
? 'initial'
DATA BASE PROCESSING COMPLETED

Figure 5-10. Program TRANPRG (Sheet 5 of 5)
### File Description With Corresponding Read Statements

A group of data items occurring NUM times

<table>
<thead>
<tr>
<th>TRANID</th>
<th>INSTID</th>
<th>INMJR</th>
<th>NUM</th>
<th>IDENT</th>
<th>COURS</th>
<th>GRADE</th>
<th>CODE</th>
<th>DATE</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR 10</td>
<td>CHAR 11</td>
<td>CHAR 20</td>
<td>INT 1</td>
<td>CHAR 14</td>
<td>CHAR 6</td>
<td>REAL 3</td>
<td>CHAR 1</td>
<td>CHAR 8</td>
<td>INT 1</td>
</tr>
</tbody>
</table>

```
READ (5,*,ERR=45,END=55)
TRANID,INSTID,INMJR,NUM

READ (5,*,ERR=45,END=45)
IDENT,COURS,GRADE,CODE,DATE,UNITS
```

### Data

- 'A000000001' '100-22-5860' 'PSYCHOLOGY' 3
- '100-22-5860-05' 'PSY136' 4.0 'C' '9/30/80' 3
- '100-22-5860-06' 'BUS001' 3.0 'C' '9/30/80' 3
- '100-22-5860-07' 'PSY003' 3.5 'C' '9/30/80' 3
- 'A000000002' '122-13-6704' 'BUSINESS' 2
- '122-13-6704-03' 'BUS017' 4.0 'C' '9/30/80' 3
- '122-13-6704-04' 'PSY003' 3.0 'C' '9/30/80' 3
- 'A000000003' '687-14-2100' 'BIOLOGY' 2
- '687-14-2100-05' 'PSY002' 4.0 'C' '9/30/80' 3
- '687-14-2100-06' 'BUS017' 3.5 'C' '9/30/80' 3

Figure 5-11. Input File INTRAN and File Description for Program TRANPRG
PROGRAM INDAVG

C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES
C INDIVIDUAL AVERAGES. THE FIRST READ SAVES THE FIRST
C STUDENT ID. WHEN THE STUDENT ID CHANGES, THE PROGRAM
C CALCULATES AND PRINTS THE AVERAGE, THEN CONTINUES TO
C READ THE FILE UNTIL EOF IS SENSED. TESTING THE THIRD
C WORD OF STATBLK CHECKS FOR EOF.
C
INTEGER STATBLK(11)
CHARACTER FINAL*3
CHARACTER OLID*11
SUBSHEMA(AVERAGE)
FINAL='NO'
INVOKE
CALL DMLBST(STATBLK,11)
PRIVACY(CFILE,PRIVACY='XX99')
OPEN(CFILE,ERR=50)
READ(CFILE,ERR=50)
IF(STATBLK(3),EQ,0,100) GO TO 40
10 N=0
TOTAL=0.0
OLID=STUDENT
20 IF(GRADE,GE,1.0) THEN
N=N+1
TOTAL=TOTAL+GRADE
ENDIF
READ(CFILE,ERR=50)
IF(STATBLK(3),EQ,0,100) THEN
FINAL='YES'
ELSE
IF(STUDENT,EQ,OLID) GO TO 20
ENDIF
IF(N,EQ,0) GO TO 30
TOTAL=TOTAL/N
PRINT 25,OLID,TOTAL
25 FORMAT ("0,9X,11,3X,F4.1)
30 IF(FINAL,EQ,'NO') GO TO 10
GO TO 100
40 PRINT *, 'EMPTY INPUT FILE'
50 PRINT 60,STATBLK
60 FORMAT (1X,'STATUS BLOCK' /
C 1 1X,04,2X,15,2X,03,2X,12,2X,
1 A10,2X,15,2X,15,2X,15,2X,A50)
100 CLOSE(CFILE)
TERMINATE
END

End-of-record
Compilation Listing

PROGRAM INDAGUE

1   PROGRAM INDAGUE
2   C
3   C THIS PROGRAM READS ALL STUDENT GRADES AND CALCULATES
4   C INDIVIDUAL AVERAGES. THE FIRST READ SAVES THE FIRST
5   C STUDENT ID. WHEN THE STUDENT ID CHANGES, THE PROGRAM
6   C CALCULATES AND PRINTS THE AVERAGE, THEN CONTINUES TO
7   C READ THE FILE UNTIL EOF IS SENSED. TESTING THE THIRD
8   C WORD OF STATBLK CHECKS FOR EOF.
9   C
10   INTEGER STATBLK(11)
11   CHARACTER FINAL*3
12   CHARACTER OLDD=11
13   ** SUBSCHEMA(AVERAGE)
14   C$ LIST(ALL=0)
15   C$ LIST(ALL)
16   FINAL='NO'
17   ** INVOKE
18   C$ LIST(ALL=0)
19   C$ LIST(ALL)
20   CALL DMLINV(0001,0BF001,10HVERAGE,10H
21       +10H,
22       +"35514310376143061021")
23   CALL DMLBLS(TATBLK,11)
24   ** PRIVACY(CFILE,PRIVACY='XX99')
25   CALL DMLPSV(1,0,0001,
26       +0"60","XX99","",""")
27   ** OPEN(CFILE,ERR=50)
28   ** READ(CFILE,ERR=50)
29   CALL DMLRD(0BF001,0001,1,0,*50)
30   IF(STATBLK(3).EQ.0"100") GO TO 40
31   N=0
32   TOTAL=0.0
33   OLDD=STUDENT
34   20 IF(GRADE .GE. 1.0) THEN
35       N=N+1
36       TOTAL=TOTAL+GRADE
37       ENDIF
38   ** READ(CFILE,ERR=50)
39   CALL DMLRD(0BF001,0001,1,0,*50)
40   IF(STATBLK(3).EQ.0"100") THEN
41       FINAL='YES'
42   ELSE
43       IF(STUDENT .EQ. OLDD) GO TO 20
44       ENDIF
45   IF(N .EQ. 0) GO TO 30
46   TOTAL=TOTAL / N
```
92 PRINT 25,OLDID,TOTAL
93 25 FORMAT ("0",9X,A11,3X,F4.1)
94 30 IF(FINAL.EQ.,'NO') GO TO 10
95 GO TO 100
96 40 PRINT *, 'EMPTY INPUT FILE'
97 50 PRINT 60,STATBLK
98 60 FORMAT (1X,'STATUS BLOCK' /
99 1 X,04,2X,15,2X,03,2X,12,2X,
100 1 A10,2X,15,2X,15,2X,15,2X,A50)
101 ** CLOSE(CFILE)
102 100 CALL DMLCLS(DBFO001,0001)
103 ** TERMINATE
104 CALL DMLEND
105 END

--VARIABLE MAP--(L0=A)
-NAME------ADDRESS------BLOCK------PROPERTIES------TYPE------SIZE
  -NAME------ADDRESS------BLOCK------PROPERTIES------TYPE------SIZE
  COURSE    2B /DB0001/    CHAR*6    DBFO001 60B /DB0000/    INTEGER   2
  DBFO001   15B /DB0000/    INTEGER  35    FINAL  314B    CHAR*3
  DBIO001   0B /DB0001/    EQV    1     GRADE  0B /DB0001AA/    REAL   14
  DBREALM   0B /DB0000/    CHAR*1    1      IDENT  0B /DB0001/    EQV    11
  DBRELST   10B /DB0000/    INTEGER  1      N   317B    INTEGER   1
  DBRID    7B /DB0000/    INTEGER  3      OLDID  315B    CHAR*11
  DBRO001   11B /DB0000/    INTEGER  3      STATBLK 301B    INTEGER   1
  DBSCNAM   4B /DB0000/    INTEGER  3      STUDENT  1B /DB0001/    CHAR*11
  DBSTAT    3B /DB0000/    INTEGER  3      TOTAL  320B    REAL   11
  DBSO001   14B /DB0000/    INTEGER

--PROCEDURES--(L0=O)
-NAME------TYPE------ARGS------CLASS------
  -NAME------TYPE------ARGS------CLASS------
  DMLCLS    2    SUBROUTINE    DMLCLS    2    SUBROUTINE
  DMLDST    2    SUBROUTINE    DMLDST    8    SUBROUTINE
  DMLEND    0    SUBROUTINE    DMLEND    5    SUBROUTINE
  DMLINV    6    SUBROUTINE    DMLINV    1    INTRINSIC

--STATEMENT LABELS--(L0=O)
-LABEL--ADDRESS------PROPERTIES------DEF
  -LABEL--ADDRESS------PROPERTIES------DEF
  10  50B    76    40    120B    96
  20  55B    79    50    122B    97
  25  154B    93    60    157B    98
  30  114B    94    100    124B    102

Figure 5-6. Program INDAVG (Sheet 3 of 4)
```
---ENTRY POINTS--- (LO=A)
-NAME---ADDRESS---ARGS---

INDAVGE 5B 0

---STATISTICS---

PROGRAM-UNIT LENGTH 321B = 209
SOM LABELLED COMMON LENGTH 67B = 55
SOM STORAGE USED 60700B = 25024
COMPILE TIME 0.052 SECONDS

Output From Program Execution

100-22-5860 3.5
120-44-3760 3.1
122-13-6704 4.0
124-33-5780 3.8
197-11-2140 3.6
437-56-8943 3.3
120-44-3760 3.5
553-89-2021 3.8
678-12-1144 4.0
687-14-2100 3.9

Figure 5-6. Program INDAVG (Sheet 4 of 4)
Control Statements

Job statement
USER statement
CHARGE statement
ATTACH(SSLIB)
DML(SB=SSLIB, LV=F5)
FTNS(I=DMLOUT)
LDSET(LIB=DMSSLIB)
LSG.
End-of-record

Source Program

PROGRAM RELATE
C
C THIS PROGRAM READS BY RELATION AND PRINTS GRADES FOR
C STUDENT ID 120-44-3760. THE FIRST READ IS RANDOM
C FOLLOWED BY A TEST FOR NULL RECORD. THE SECOND READ IS
C SEQUENTIAL FOLLOWED BY A TEST FOR CONTROL BREAK.
C
INTEGER STATBLK(11)
CHARACTER SAVEKEY*11
SUBSCHEMA(RELATION)
INVOK
CALL DMLDBST(STATBLK,11)
PRIVACY(CFILE, PRIVACY='XX99')
OPEN(REL1, ERR=50)
STID='120-44-3760'
10 READ(REL1, KEY=STID, ERR=50)
SAVEKEY=STID
IF(SAVEKEY.EQ.'JJJJJJJJJJ') GO TO 70
15 PRINT 20, CSTID, GRADE
20 FORMAT (1H0, 9X, A11, 3X, F4.1)
25 READ(REL1, ERR=50, END=70)
IF(STATBLK(7).NE.0) GO TO 70
GO TO 15
50 PRINT 60, STATBLK
60 FORMAT (1X, 'STATUS BLOCK' /
1 1X, 04, 2X, 15, 2X, 03, 2X, 12, 2X,
1 A10, 2X, 15, 2X, 15, 2X, 15, 2X, A30)
70 CLOSE(REL1)
TERMINATE
END
End-of-record

Figure 5-7. Program RELATE (Sheet 1 of 4)
Compilation Listing

PROGRAM RELATE 76/176 OPT=0 F7N 5.1+538

1    PROGRAM RELATE
2      C
3      C THIS PROGRAM READS BY RELATION AND PRINTS GRADES FOR
4      C STUDENT ID 120-44-3760. THE FIRST READ IS RANDOM
5      C FOLLOWED BY A TEST FOR NULL RECORD. THE SECOND READ IS
6      C SEQUENTIAL FOLLOWED BY A TEST FOR CONTROL BREAK.
7      C
8      INTEGER STATBLK(11)
9      CHARACTER SAVEKEY*11
10     ** SUBSCHEMA(RELATION)
11     CS LIST(ALL=0)
12     CS LIST(ALL)
13     ** INVOKE
14     CS LIST(ALL)
15     CALL DMLINV(DBFDO01,10HRELATION ,10H
16          +10H  ,0"76710464332261536703"")
17     CALL DMLBST(STATBLK,11)
18     ** PRIVACY(CFILE,PRIVACY='XX99')
19     CALL DMLPRV(1,1,0,0002,
20          +0"60","XX99"
21          +")
22     ** OPEN(REL1,ERR=50)
23     CALL DMLPWT(DBN0001,DBA0001,ZH10,*,50)
24     STID=120-44-3760
25     ** READ(REL1,KEY=STID,ERR=50)
26     10 CALL DMLRLK(DBN0001,0001,00001,0001,1,0011,0,0000,00,STID
27            + 0001,*,50)
28        SAVEKEY=STID
29        IF(SAVEKEY.EQ.'3333333333') GO TO 70
30     15 PRINT 20, CSTID, GRADE
31     20 FORMAT (1H0,9X,A11,5X,F4.1)
32     ** READ(REL1,ERR=50,END=70)
33     25 CALL DMLRL(DBN0001,0001,1,1,*,50 ,*,70)
34        IF(STATBLK(7).NE.0) GO TO 70
35     29 GO TO 15
36     50 PRINT 60,STATBLK
37     60 FORMAT (1X,'STATUS BLOCK' /
38           1 1X,04,2X,15,2X,03,2X,12,2X,1
39           1 A10,2X,15,2X,15,2X,15,2X,A30)
40     ** CLOSE(REL1)
41     70 CALL DMLCLS(DBN0001,DBA0001)
42     ** TERMINATE
43     77 CALL DMLEND
44     END

Figure 5-7. Program RELATE (Sheet 2 of 4)
Control Statements

Job statement
USER statement
CHANGE statement
ATTACH(SSLIB)
DML(SB=SSLIB, LV=F5)
FTNS(I=DMLOUT)
LDSET(LIB=DMLIB)
LGO.
End-of-record

Source Program

PROGRAM CHARGES

C THIS PROGRAM DEMONSTRATES SUBSCRIPTING AND THE DML
C REWRITE CAPABILITY. RELATION REL2 ASSOCIATES FILES
C STUDENT AND ACCOUNT. FILE ACCOUNT IS DEFINED IN THE
C SCHEMA AS HAVING REPEATING ITEMS. THIS PROGRAM PERFORMS A
C RANDOM READ TO SELECT STUDENT ID 197-11-2140 AND
C WRITE REAL VALUES INTO THREE FIELDS. A TOTAL IS CALCULATED
C AND PRINTED OUT ALONG WITH THE NAME OF THE STUDENT.
C
INTEGER STATBLK(11)
SUBSCHEMA(BURSAR)
INVOKESDML(RSTBLK,11)
TOTAL=0
OPEN(REL2,ERR=70)
STD='197-11-2140'
READ(REL2,KEY=STD,ERR=70)
TUITION(1)=1400
LAB(1)=75
BOOKS(1)=146
REWRITE(ACCOUNT, ERR=70)
TOTAL=TUITION(1) + LAB(1) + BOOKS(1)
50 PRINT 60,NAME,TOTAL
50 FORMAT("0",9X,A30,3X,F8.2)
GO TO 90
70 PRINT 80,STATBLK
80 FORMAT (1X,'STATUS BLOCK' ,/1
1 1X,04,2X,A5,2X,03,2X,12,2X,
1 A10,2X,A5,2X,A5,2X,A5,2X,A30)
90 CLOSE(REL2)
TERMINATE
END
End-of-record

Figure 5-8. Program CHARGES (Sheet 1 of 3)
Compilation Listing

PROGRAM CHARGES  76/176  OPT=0  

1 PROGRAM CHARGES
2 C
3 C THIS PROGRAM DEMONSTRATES SUBSCRIPITNG AND THE DML
4 C REWRITE CAPABILITY. RELATION REL2 ASSOCIATES FILES
5 C STUDENT AND ACCOUNT. FILE ACCOUNT IS DEFINED IN THE
6 C SCHEMA AS HAVING REPEATING ITEMS. THIS PROGRAM PERFORMS A
7 C RANDOM READ TO SELECT STUDENT ID 197-11-2140 AND
8 C WRITE REAL VALUES INTO THREE FIELDS. A TOTAL IS CALCULATED
9 C AND PRINTED OUT ALONG WITH THE NAME OF THE STUDENT.
10 C
11 INTEGER STATBLK(11)
12 ** SUBSCHEMA(BURSAR)
13 C$ LIST(ALL=0)
14 C$ LIST(ALL)
15 ** INVOKE
16 C$ LIST(ALL)
17 C$ LIST(ALL)
18 CALL DMLINV(0002, DBF0001, 10HBURSAR, 10H
19 + 10H
20 CALL DMLDSBT(STATBLK, 11)
21 TOTAL=0
22 ** OPEN(REL2, ERR=70)
23 CALL DMLOPN(0BN0001, DBA0001, 2HIO, 70 )
24 ** READ(REL2, KEY=STD, ERR=70)
25 ** CALL DMLRLK(0BN0001, 0001, 0001, 0001, 1, 0011, 0, 0000, 00, STD
26 + 001, 70 )
27 TUITION(1)=1400
28 LAB(1)=75
29 BOOKS(1)=146
30 ** REWRITE(ACCOUNT, ERR=70)
31 CALL DMLREW(DBF0002, 0, 0002, 00001, *70 )
32 ** TOTAL=TUITION(1) + LAB(1) + BOOKS(1)
33 50 PRINT 60, NAME, TOTAL
34 60 FORMAT("0", 9X, A30, 3X, F8.2)
35 GO TO 90
36 70 PRINT 80, STATBLK
37 80 FORMAT (1X, 'STATUS BLOCK', /
38 1 1X, 04, 2X, I5, 2X, 03, 2X, I2, 2X,
39 1 A10, 2X, I5, 2X, I5, 2X, I5, 2X, A30)
40 ** CLOSE(REL2)
41 CALL DMLCLSR(0BN0001, DBA0001)
42 ** TERMINATE
43 CALL DMLEND
44 END

Figure 5-8. Program CHARGES (Sheet 2 of 3)
---VARIABLE MAP---(LO=A)
-NAME----ADDRESS----BLOCK----PROPERTIES----TYPE----SIZE

| ADDR     | 4B   | /D80001/ | CHAR*20 | DB10001 | 0B  | /D80001/ | EQU | CHAR*1  |
| ASTID    | 0B   | /D80002/ | CHAR*11 | DB10002 | 0B  | /D80002/ | EQU | CHAR*1  |
| BOOKS    | 40B  | /D0002AA/| REAL    | 16      | DBN001 | 134B | /D80000/ | INTEGER | 3     |
| CITY     | 6B   | /D80001/ | CHAR*10 | DBREALM | 0B  | /D80000/ | INTEGER | 3     |
| DBA0001  | 255B |         | INTEGER | 3       | DBRELST | 108  | /D80000/ | INTEGER | 2     |
| DBFO001  | 168  | /D80000/ | INTEGER | 35      | DBRUID  | 78   | /D80000/ | INTEGER | 3     |
| DBFO002  | 67B  | /D80000/ | INTEGER | 35      | DBRO001 | 128  | /D80000/ | INTEGER | 3     |
| DBFO003  | 63B  | /D80000/ | INTEGER | 3       | MISCO   | 608  | /D0002AA/| REAL    | 16    |
| DBSCAM   | 24B  | /D80000/ | INTEGER | 3       | NAME    | 18   | /D80001/ | CHAR*30 |       |
| DBSTAT   | 1B   | /D80000/ | INTEGER | 2       | TOTAL   | 260B | /D80000/ | REAL    |       |
| DBSO001  | 16B  | /D80000/ | INTEGER | 2       | TUITION | 78   | /D80001/ | CHAR*2  |       |
| DBSO002  | 66B  | /D80000/ | INTEGER | 2       | ZIP     | 78   | /D80001/ | CHAR*11 |       |
| DBO0001  | 66B  | /D80000/ | INTEGER | 2       | REAL    | 16   | /D80001/ | CHAR*5  |       |

---PROCEDURES---(LO=A)
-NAME--------TYPE-------ARGS------CLASS------

| DMLCSR    | 2     | SUBROUTINE | DMLOPN    | 4     | SUBROUTINE |
| DMLDSB    | 2     | SUBROUTINE | DMLREW    | 5     | SUBROUTINE |
| DMLSNB    | 0     | SUBROUTINE | DMLBLK    | 12    | SUBROUTINE |
| DMLINV    | 6     | SUBROUTINE | LOC5     GENERIC | 1 INTRINSIC |

---STATEMENT LABELS---(LO=A)
-LABEL-ADDRESS-----PROPERTIES-----DEF

| 50 *NO REFS* | 133 |
| 60            | 135B | FORMAT | 134 |
| 70            | 105B |        | 136 |
| 80            | 140B | FORMAT | 137 |
| 90            | 107B |        | 141 |

---ENTRY POINTS---(LO=A)
-NAME----ADDRESS----ARGS-----

| CHARGES | 5B   | 0     |

---STATISTICS---

-PROGRAM-UNIT LENGTH 261B = 177  
-SCM LABELLED COMMON LENGTH 251B = 169  
-SCM STORAGE USED 60700B = 25024  
-COMPILE TIME 0.056 SECONDS

Output From Program Execution
CAREN NIELSON 1621.00

Figure 5-8. Program CHARGES (Sheet 3 of 3)
Control Statements

Job statement
USER statement
CHANGE statement
ATTACH(SSTLIB)
DML(SB=SSTLIB,LV=05)
FINS(I=DMLOUT)
LDSET(LIB=DSTLIB)
LGO.
End-of-record

Source Program

PROGRAM ADMIT
C
C THIS PROGRAM DEMONSTRATES THE USE OF SUBROUTINES.
C WHEN AN INCOMPLETE(I) CODE IS FOUND IN FILE CFIL.
C SUBROUTINE PRNTSUB IS CALLED TO PRINT THE STUDENT ID
C AND COURSE ID. NOTICE THAT THE SUBROUTINE REQUIRES
C A SUBSCHEMA AND INVOKE STATEMENT. THESE STATEMENTS
C ARE REQUIRED IN EVERY PROGRAM UNIT TO ENSURE THE SAME
C DATA IS BEING REFERENCED IN COMMON.
C
INTEGER STATBLK(11)
SUBSCHEMA(ADMISSIONS)
INVOC
CALL DMLDBST(STATBLK,11)
PRIVACY(CFIL,PRIVCY='XX99')
OPEN(CFIL,ERR=70)
10 READ(CFIL,ERR=70,END=100)
IF(CODE .EQ. 'I') CALL PRNTSUB
GO TO 10
70 PRINT 80,STATBLK
80 FORMAT (1X,'STATUS BLOCK' /
1 1X,04,2X,I5,2X,05,2X,I2,2X,
1 A10,2X,I5,2X,I5,2X,A50)
100 CLOSE(CFIL)
TERMINATE
END
C
SUBROUTINE PRNTSUB
SUBSCHEMA(ADMISSIONS)
INVOC
PRINT 50,STUDENT,CCID
50 FORMAT("O",9X,A11,5X,A6)
RETURN
END
End-of-record

Figure 5-9. Program ADMIT (Sheet 1 of 5)
Compilation Listing

PROGRAM ADMIT  76/176  OPT=0  FTN 5.1+538

1     PROGRAM ADMIT
2     C
3     C THIS PROGRAM DEMONSTRATES THE USE OF SUBROUTINES.
4     C WHEN AN INCOMPLETE() CODE IS FOUND IN FILE CFILE,
5     C SUBROUTINE PRNTSUB IS CALLED TO PRINT THE STUDENT ID
6     C AND COURSE ID. NOTICE THAT THE SUBROUTINE REQUIRE
7     C A SUBSCHEMA AND INVOKE STATEMENT. THESE STATEMENTS
8     C ARE REQUIRED IN EVERY PROGRAM UNIT TO ENSURE THE SAME
9     C DATA IS BEING REFERENCED IN COMMON.
10     C
11     INTEGER STATBLK(11)
12     ** SUBSCHEMA(ADMISSIONS)
13     C$ LIST(ALL=0)
14     C$ LIST(ALL)
15     ** INVOKE
16     C$ LIST(ALL=0)
17     C$ LIST(ALL)
18     CALL DMLINV(0002, DBFOOO1,10HADMISSIONS,10H
19         +10H ,"5606531337542307610")
20     CALL DMLBST(STATBLK,11)
21     ** PRIVACY(CFILE,PRIVACY='XX99')
22     CALL DMLPRV(1,1,0,0002,
23         +0"60","XX99","",""")
24     ** OPEN(CFILE,ERR=70)
25     CALL DMLOPEN(DBFOOO2,0002,2H10,*70 )
26     ** READ(CFILE,ERR=70,END=100)
27     10 CALL DMLRD(DBFOOO2,0002,1,1,*70 ,#100 )
28     IF(CHAR, .EQ. '1') CALL PRNTSUB
29     GO TO 10
30     70 PRINT 80,STATBLK
31     80 FORMAT (1X,'STATUS BLOCK' /
32         1 1X,04,2X,15,2X,03,2X,12,2X,
33         1 A10,2X,15,2X,15,2X,15,2X,A30)
34     ** CLOSE(CFILE)
35     100 CALL DMLCLS(DBFOOO2,0002)
36     ** TERMINATE
37     116 CALL DMLEND
38     END

Figure 5-9. Program ADMIT (Sheet 2 of 5)
---VARIABLE MAP---(L=40)
-NAME--ADDRESS--BLOCK--PROPERTIES--TYPE--SIZE
CCID   2B /DB0002/   CHAR*6
C1D    08 /DB0001/   CHAR*6
CODE   3B /DB0002/   CHAR*1
DBFOO1 15B /DB0002/   INTEGER 35
DBFO02 66B /DB0002/   INTEGER 35
DB1000 08 /DB0001/   CHAR*1
DB2000 08 /DB0002/   CHAR*1
DBREALM 08 /DB0000/   INTEGER 3
DBREST 10B /DB0000/   INTEGER 1
DBRUID 7B /DB0000/   INTEGER 3
DBRO002 11B /DB0000/   INTEGER 3
DBSCNAM 4B /DB0000/   INTEGER 3
DBSTAT 3B /DB0000/   INTEGER
DBSO00 14B /DB0000/   INTEGER
DBSO02 65B /DB0000/   INTEGER 2
DBT000 60B /DB0000/   INTEGER 2
DBT002 13B /DB0000/   INTEGER 2
IDENT  08 /DB0002/   EQV
NAME   08 /DB0001/   CHAR*20
PREREG 4B /DB0001/   CHAR*6
SCHOOL 2B /DB0001/   CHAR*20
STATBLK 17B /DB0000/   CHAR*11
STUDENT 1B /DB0002/   CHAR*6
UNITS   08 /DB001AA/   INTEGER

---PROCEDURES---(L=40)
-NAME--TYPE--ARGS--CLASS
DMLCLS 2 SUBROUTINE
DMLDBST 2 SUBROUTINE
DMLEND 0 SUBROUTINE
DMLINV 6 SUBROUTINE
DMLNOPN 4 SUBROUTINE
DMLPRV 8 SUBROUTINE
DMLRD 6 SUBROUTINE
LOCF GENERIC 1 INTRINSIC
PRNTSUB 0 SUBROUTINE

---STATEMENT LABELS---(L=40)
-LABEL--ADDRESS--PROPERTIES--DEF
10 44B 106
70 60B 109
80 106B FORMAT 110
100 62B 114

---ENTRY POINTS---(L=40)
-NAME--ADDRESS--ARGS--
ADMIT 58 0

---STATISTICS---
PROGRAM-UNIT LENGTH   212B = 138
SCM LABELLED COMMON LENGTH 146B = 102
SCM STORAGE USED 60700B = 25024
COMPILE TIME 0.046 SECONDS

Figure 5-9. Program ADMIT (Sheet 3 of 5)
SUBROUTINE PRNTSUB 76/176 OPT=0  FTN 5.1+538

1 C
2 ** SUBROUTINE PRNTSUB
3 SUBSCHEMA(ADMISSIONS)
4 C$ LIST(ALL=0)
5 C$ LIST(ALL)
6 ** INVOKE
7 C$ LIST(ALL=0)
8 C$ LIST(ALL)
9 CALL DMLINV(OO002,0BF0001,10HADMISSIONS,10H
10 +10H "0"5606531377542307610")
11 PRINT 50,STUDENT,CCID
12 50 FORMAT("D",9X,A11,3X,A6)
13 RETURN
14 END

---VARIABLE MAP---(LO=A)
-NAME---ADDRESS---BLOCK-----PROPERTIES-------TYPE--------SIZE
CCID  2B /DB0002/ CHAR*6 DBSCNAME  4B /DB0000/ INTEGER  3
CID   0B /DB0001/ EQV CHAR*6 DBSTAT   3B /DB0000/ INTEGER
CODE  3B /DB0002/ CHAR*1 DBSO001  14B /DB0000/ INTEGER
DBF0001 15B /DB0000/ INTEGER  35 DBTO002  65B /DB0000/ INTEGER
DBF0002 66B /DB0000/ INTEGER  35 DBTO001  60B /DB0000/ INTEGER
DB10001 0B /DB0001/ EQV CHAR*1 DBTO002 131B /DB0000/ INTEGER
DB10002 0B /DB0002/ EQV CHAR*1 IDENT   0B /DB0002/ EqV CHAR*14
DBREALM 0B /DB0000/ INTEGER  3 NAME    0B /DB0001/ CHAR*20
DBRELST 10B /DB0000/ INTEGER  1 PREREQ   4B /DB0001/ CHAR*6
DBRU1D  7B /DB0000/ INTEGER  1 SCHOOL   2B /DB0001/ CHAR*20
DBR0001 11B /DB0000/ INTEGER  3 STUDENT  1B /DB0002/ CHAR*11
DBR0002 62B /DB0000/ INTEGER  3 UNITS    0B /DB001AA/ INTEGER

---PROCEDURES---(LO=A)
-NAME------TYPE------ARGS------CLASS------
DMLINV  GENERIC  6 SUBROUTINE
LOCF   GENERIC  1 INTRINSIC

---STATEMENT LABELS---(LO=A)
-LABEL-ADDRESS-----PROPERTIES-----DEF
50  44B FORMAT 91

Figure 5-9. Program ADMIT (Sheet 4 of 5)
--ENTRY POINTS--(LO=A)
-NAME--ADDRESS--ARGS--
PRNTSUB 48 0

--STATISTICS--

PROGRAM-UNIT LENGTH 67B = 55
SCM LABELLED COMMON LENGTH 146B = 102
SCM STORAGE USED 607009 = 25024
COMPILE TIME 0.033 SECONDS

Output From Program Execution

100-22-5860 PSY002
122-13-6704 PSY136
387-14-1232 MATH10
387-14-1232 CHM005
387-14-1232 PSY136
387-14-1232 BUS017
678-12-1144 HIS103

Figure 5-9. Program ADMIT (Sheet 5 of 5)
This user's guide assumes that CDCS is active and available for your job. This method of operation implies established schemas, appropriate sub-schema libraries, and successfully implemented applications. Any change in the data base environment, such as the addition of a new file definition, forces the data administrator to terminate CDCS and to reinitiate CDCS with a new master directory file attached. By using the CDCS Batch Test Facility, you can have CDCS running as a normal batch job. Each time you run your job, you attach a new version of the master directory file.

The CDCS Batch Test Facility is an absolute program called CDCSBTF. The program resides on the system library and is called into execution by the CDCSBTF control statement. The format and parameters of the CDCSBTF control statement are shown in figure 6-1. The CDCSBTF control statement cannot exceed 80 characters in length. A slash (/) indicates the end of the list of user program file names and the beginning of the parameter list.

When CDCS is executing as the Batch Test Facility, the name of the output file produced by CDCS is CDCSOUT.

DIRECTIVE FILE

An optional directive file can be used to contain the parameters in addition to or instead of the parameters in the CDCSBTF control statement. Using a directive file allows specification of a parameter list that is longer than that allowed in the control statement. With the exception of the MFL parameter, which cannot be specified in either the control statement or the directive file. The same parameter cannot, however, be specified in both the control statement and the directive file. Any of the parameters, except MFL or DIR, that are valid in the CDCSBTF control statement are also valid in the directive file. Parameters can be specified in the directive file in columns 1 through 80. The first parameter specified in a line must begin in column 1. Parameters can either be placed in separate lines or combined in a line with commas acting as separators. Parameters cannot be split across lines.

PARAMETERS

All the parameters (figure 6-1) of the CDCSBTF control statement are optional and can be specified in any order. The CDCSBTF parameters provide information for the following functions:

- Allocation of maximum pooled buffer space

CDCSBTF(lfn-1 lfn-2 ... lfn-n)

| lfn | Specifies the logical file name of a relocatable binary file containing a user program. Up to 16 files can be specified. |
| p | A parameter; the parameters are as follows: |
| DIR=lfn | Directive file for CDCSBTF control statement parameters |
| BL=nn | Maximum pooled buffer space |
| CP=t1 | Central processor time |
| IO=t2 | Input/output time |
| MFL=fl | Maximum field length for CDCSBTF |
| CRM=fs1/fs2/... | Load CRM capsules where fs is a file structure as follows: |
| MDPFN=pfn | Permanent file information for master directory file |
| UN=user-name | |
| ID=user-name | |
| PW=pwrd1/pwrd2/... | |
| FAM=family-name | |
| PN=pack-name | |
| DT=device-set | |
| SN=set-name | |

Figure 6-1. CDCSBTF Control Statement Format

- Adjustment of accounting charges
- Allocation of the maximum field length that CDCSBTF is allowed to use
- Specification of information required to attach the master directory; these parameters must be given if online dumping of journal log files is desired
- Specification of a directive file that can contain any of the CDCSBTF control statement parameters except MFL and DIR
If the CP and I0 parameters are specified either in the CDCSBTF control statement or in the directive file, the accounting values return to the user's dayfile are different from the accounting values returned when the same application executes with CDCS at the system control point. When CDCSBTF executes, the accounting values returned include the application's execution time as well as the central processor and input/output time charged by CDCS.

The parameters available for the CDCSBTF control statement are the same as the parameters available for the CDCS control statement. See the CDCS Data Administrator's reference manual for more information about the parameters.

**REQUIREMENTS**

When you are running application programs with the CDCSBTF program, you must meet certain requirements. These requirements are:

- Attach the master directory file. It can be attached in two ways: either by information provided in the CDCSBTF control statement or directive file or by an ATTACH control statement that precedes execution of CDCSBTF. If the ATTACH control statement is used, the master directory file must be specified with the local file name MSTRDIR. However, using this method of attaching the master directory prevents online dumping of journal log files from being performed by CDCSBTF.

- Have the application program in relocatable binary format as a local or a permanent file.

- Assign unique names to non-CDCS files. Do not use any of the following names:

  INPUT
  OUTPUT
  MSTRDIR
  CDCSOUT
  Fnnnnnn
  Jnnnnnn
  Pnnnnnn
  Qnnnnnn
  Rnnnnnn
  Tnnnnnn
  Xnnnnnn
  A name beginning with ZZZZZ
  A log file name

- Be sure your application program executes a DML TERMINATE statement before a FORTRAN STOP or END statement. Failure to do this would discontinue processing for all programs specified in the CDCSBTF control statement that have not completed execution.

- Set the DB parameter in the FTN5 control statement equal to 0. Multiple copies of CDCSBTF can be run concurrently, and as many as 16 user programs can be run with a copy of CDCSBTF. If more than two concurrent calls to the RECOVR routine are made, CDCS aborts processing.

**OBTAINING LOAD MAPS**

You can obtain load maps by setting sense switches 1 through 4 prior to execution of the CDCSBTF control statement. Each sense switch setting corresponds to different information on the load map. The settings and the associated types of information are shown in table 6-1.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Load Map Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH(1)</td>
<td>Statistics (S)</td>
</tr>
<tr>
<td>SWITCH(2)</td>
<td>Block maps (B)</td>
</tr>
<tr>
<td>SWITCH(3)</td>
<td>Entry point maps (EO)</td>
</tr>
<tr>
<td>SWITCH(4)</td>
<td>Entry point cross-reference maps (X).</td>
</tr>
</tbody>
</table>

**EXECUTING THE CDCS BATCH TEST FACILITY**

You need to include a number of control statements when executing the CDCS Batch Test Facility for your FORTRAN application program. Figure 6-2 provides a sample list of statements. Parameters correspond to the application that appears in appendix C.
<table>
<thead>
<tr>
<th>NOS Operating System</th>
<th>NOS/BE Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobname,CMfl.</td>
<td>Jobname,CMfl.</td>
</tr>
<tr>
<td>USER statement</td>
<td></td>
</tr>
<tr>
<td>CHARGE statement</td>
<td></td>
</tr>
<tr>
<td>ATTACH,SSLIB/UN=xxx.</td>
<td>ATTACH,SSLIB,ID=xxx.</td>
</tr>
<tr>
<td>DML,SB=SSLIB,lv=F5</td>
<td>DML,SB=SSLIB,lv=F5</td>
</tr>
<tr>
<td>FTN5,I=DMLOUT,DB=0.</td>
<td>FTN5,I=DMLOUT,DB=0.</td>
</tr>
<tr>
<td>SWITCH,2.</td>
<td>SWITCH,2.</td>
</tr>
<tr>
<td>SWITCH,3.</td>
<td>SWITCH,3.</td>
</tr>
<tr>
<td>SWITCH,4.</td>
<td>SWITCH,4.</td>
</tr>
<tr>
<td>LIBRARY,DMSLIB.</td>
<td>LIBRARY,DMSLIB.</td>
</tr>
<tr>
<td>CDCSBTF,LAGO/MDFN=MSTRDIR,UN=xxx.</td>
<td>CDSBF,LAGO/MDFN=MSTRDIR,ID=xxx.</td>
</tr>
<tr>
<td>REWIND,CDCSOUT.</td>
<td>REWIND,CDCSOUT.</td>
</tr>
<tr>
<td>COPY,CDCSOUT,OUTPUT.</td>
<td>COPY,CDCSOUT,OUTPUT.</td>
</tr>
<tr>
<td>CRMEP.</td>
<td>CRMEP.</td>
</tr>
<tr>
<td>MD,M=377000.</td>
<td>M=377000.</td>
</tr>
<tr>
<td>CRMEP.</td>
<td>CRMEP.</td>
</tr>
<tr>
<td>REWIND,CDCSOUT.</td>
<td>REWIND,CDCSOUT.</td>
</tr>
<tr>
<td>COPY,CDCSOUT,OUTPUT.</td>
<td>COPY,CDCSOUT,OUTPUT.</td>
</tr>
</tbody>
</table>

Names the job and specifies maximum field length.
Identifies the user.
Specifies the account to which the job's use of system resources is logged.
Attaches the sub-schema.
Preprocesses the DML statements in the FORTRAN program and writes to DMLOUT.
Compiles the FORTRAN program on DMLOUT and places it on the LGO file.
Requests block map on program-initiated load.
Requests entry point map on program-initiated load.
Requests entry point cross reference map on program-initiated load.
Specifies that library DMSLIB is to be used to satisfy externals.
Executes CDCSBTF and passes the master directory permanent file information as parameters.
Rewinds the CDCSBTF output file.
Prints the CDCSBTF output file.
Prints the CRM error file.
Establishes processing if error occurs.
Dumps the exchange package.
Dumps the contents of the field length.
Prints the CRM error file.
Rewinds the CDCSBTF output file.
Prints the CDCSBTF output file.

Figure 6-2. Sample FORTRAN 5 Execution of CDCS Batch Test Facility
<table>
<thead>
<tr>
<th>Collating Sequence</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
<th>Collating Sequence</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>blank</td>
<td>55</td>
<td>20</td>
<td>32 40</td>
<td>@</td>
<td>74</td>
<td>40</td>
</tr>
<tr>
<td>01 01</td>
<td>!</td>
<td>66</td>
<td>21</td>
<td>33 41</td>
<td>A</td>
<td>01</td>
<td>41</td>
</tr>
<tr>
<td>02 02</td>
<td>&quot;</td>
<td>64</td>
<td>22</td>
<td>34 42</td>
<td>B</td>
<td>02</td>
<td>42</td>
</tr>
<tr>
<td>03 03</td>
<td>#</td>
<td>60</td>
<td>23</td>
<td>35 43</td>
<td>C</td>
<td>03</td>
<td>43</td>
</tr>
<tr>
<td>04 04</td>
<td>$</td>
<td>53</td>
<td>24</td>
<td>36 44</td>
<td>D</td>
<td>04</td>
<td>44</td>
</tr>
<tr>
<td>05 05</td>
<td>%</td>
<td>63†</td>
<td>25</td>
<td>37 45</td>
<td>E</td>
<td>05</td>
<td>45</td>
</tr>
<tr>
<td>06 06</td>
<td>&amp;</td>
<td>67</td>
<td>26</td>
<td>38 46</td>
<td>F</td>
<td>06</td>
<td>46</td>
</tr>
<tr>
<td>07 07</td>
<td>`</td>
<td>70</td>
<td>27</td>
<td>39 47</td>
<td>G</td>
<td>07</td>
<td>47</td>
</tr>
<tr>
<td>08 10</td>
<td>(</td>
<td>51</td>
<td>28</td>
<td>40 50</td>
<td>H</td>
<td>10</td>
<td>48</td>
</tr>
<tr>
<td>09 11</td>
<td>)</td>
<td>52</td>
<td>29</td>
<td>41 51</td>
<td>I</td>
<td>11</td>
<td>49</td>
</tr>
<tr>
<td>10 12</td>
<td>*</td>
<td>47</td>
<td>2A</td>
<td>42 52</td>
<td>J</td>
<td>12</td>
<td>4A</td>
</tr>
<tr>
<td>11 13</td>
<td>+</td>
<td>45</td>
<td>2B</td>
<td>43 53</td>
<td>K</td>
<td>13</td>
<td>4B</td>
</tr>
<tr>
<td>12 14</td>
<td>.</td>
<td>56</td>
<td>2C</td>
<td>44 54</td>
<td>L</td>
<td>14</td>
<td>4C</td>
</tr>
<tr>
<td>13 15</td>
<td>-</td>
<td>46</td>
<td>2D</td>
<td>45 55</td>
<td>M</td>
<td>15</td>
<td>4D</td>
</tr>
<tr>
<td>14 16</td>
<td>.</td>
<td>57</td>
<td>2E</td>
<td>46 56</td>
<td>N</td>
<td>16</td>
<td>4E</td>
</tr>
<tr>
<td>15 17</td>
<td>/</td>
<td>50</td>
<td>2F</td>
<td>47 57</td>
<td>O</td>
<td>17</td>
<td>4F</td>
</tr>
<tr>
<td>16 20</td>
<td>0</td>
<td>33</td>
<td>30</td>
<td>48 60</td>
<td>P</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>17 21</td>
<td>1</td>
<td>34</td>
<td>31</td>
<td>49 61</td>
<td>Q</td>
<td>21</td>
<td>51</td>
</tr>
<tr>
<td>18 22</td>
<td>2</td>
<td>35</td>
<td>32</td>
<td>50 62</td>
<td>R</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>19 23</td>
<td>3</td>
<td>36</td>
<td>33</td>
<td>51 63</td>
<td>S</td>
<td>23</td>
<td>53</td>
</tr>
<tr>
<td>20 24</td>
<td>4</td>
<td>37</td>
<td>34</td>
<td>52 64</td>
<td>T</td>
<td>24</td>
<td>54</td>
</tr>
<tr>
<td>21 25</td>
<td>5</td>
<td>40</td>
<td>35</td>
<td>53 65</td>
<td>U</td>
<td>25</td>
<td>55</td>
</tr>
<tr>
<td>22 26</td>
<td>6</td>
<td>41</td>
<td>36</td>
<td>54 66</td>
<td>V</td>
<td>26</td>
<td>56</td>
</tr>
<tr>
<td>23 27</td>
<td>7</td>
<td>42</td>
<td>37</td>
<td>55 67</td>
<td>W</td>
<td>27</td>
<td>57</td>
</tr>
<tr>
<td>24 30</td>
<td>8</td>
<td>43</td>
<td>38</td>
<td>56 70</td>
<td>X</td>
<td>30</td>
<td>58</td>
</tr>
<tr>
<td>25 31</td>
<td>9</td>
<td>44</td>
<td>39</td>
<td>57 71</td>
<td>Y</td>
<td>31</td>
<td>59</td>
</tr>
<tr>
<td>26 32</td>
<td>:</td>
<td>00†</td>
<td>3A</td>
<td>58 72</td>
<td>Z</td>
<td>32</td>
<td>5A</td>
</tr>
<tr>
<td>27 33</td>
<td>;</td>
<td>77</td>
<td>3B</td>
<td>59 73</td>
<td>[</td>
<td>61</td>
<td>5B</td>
</tr>
<tr>
<td>28 34</td>
<td>&lt;</td>
<td>72</td>
<td>3C</td>
<td>60 74</td>
<td>\</td>
<td>75</td>
<td>5C</td>
</tr>
<tr>
<td>29 35</td>
<td>=</td>
<td>54</td>
<td>3D</td>
<td>61 75</td>
<td>]</td>
<td>62</td>
<td>5D</td>
</tr>
<tr>
<td>30 36</td>
<td>&gt;</td>
<td>73</td>
<td>3E</td>
<td>62 76</td>
<td>^</td>
<td>76</td>
<td>5E</td>
</tr>
<tr>
<td>31 37</td>
<td>?</td>
<td>71</td>
<td>3F</td>
<td>63 77</td>
<td>_</td>
<td>65</td>
<td>5F</td>
</tr>
</tbody>
</table>

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.
### NOS Operating System

<table>
<thead>
<tr>
<th>User Statement</th>
<th>NOS/BE Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobname,CMfl.</td>
<td>Jobname,CMfl.</td>
</tr>
<tr>
<td>USER statement</td>
<td></td>
</tr>
<tr>
<td>CHARGE statement</td>
<td></td>
</tr>
</tbody>
</table>

**ATTACH, SSLIB/UN=xxx.**

**DML, SB=SSLIB, LV=F5**

**FTN5, I=DMLOUT, DB=0.**

**SWITCH, 2.**

**SWITCH, 3.**

**SWITCH, 4.**

**LIBRARY, DMSLIB.**

**CDCSBTF, LGO/MDPFN=MDTB, UN=xxx.**

**REWIND, CDCSOUT.**

**COPY, CDCSOUT, OUTPUT.**

**CRMEP.**

**DMD.**

**DMD, 377000.**

**CRMEP.**

**REWIND, CDCSOUT.**

**COPY, CDCSOUT, OUTPUT.**

---

### NOS/BE Operating System

<table>
<thead>
<tr>
<th>User Statement</th>
<th>NOS/BE Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobname, CMfl.</td>
<td>Jobname, CMfl.</td>
</tr>
<tr>
<td>USER statement</td>
<td></td>
</tr>
<tr>
<td>CHARGE statement</td>
<td></td>
</tr>
</tbody>
</table>

**ATTACH, SSLIB, ID=xxx.**

**DML, SB=SSLIB, LV=F5**

**FTN5, I=DMLOUT, DB=0.**

**SWITCH, 2.**

**SWITCH, 3.**

**SWITCH, 4.**

**LIBRARY, DMSLIB.**

**CDCSBTF, LGO/MDPFN=MDTB, ID=xxx.**

**REWIND, CDCSOUT.**

**COPY, CDCSOUT, OUTPUT.**

**CRMEP.**

**DMD.**

**DMD, 377000.**

**CRMEP.**

**REWIND, CDCSOUT.**

**COPY, CDCSOUT, OUTPUT.**

---

**Figure 6-2. Sample FORTRAN 5 Execution of CDCS Batch Test Facility**
Control Data operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed or (for NOS only) dead started.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified also by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described previously for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Standard collating sequences for the two printer character sets are shown in tables A-2 and A-3.
## Table A-1. Standard Character Sets

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00†</td>
<td>(:colon) ‡‡</td>
<td>8-2</td>
<td>00</td>
<td>(:colon) ‡‡</td>
<td>8-2</td>
<td>072</td>
</tr>
<tr>
<td>01</td>
<td>A</td>
<td>12-1</td>
<td>61</td>
<td>A</td>
<td>12-1</td>
<td>101</td>
</tr>
<tr>
<td>02</td>
<td>B</td>
<td>12-2</td>
<td>62</td>
<td>B</td>
<td>12-2</td>
<td>102</td>
</tr>
<tr>
<td>03</td>
<td>C</td>
<td>12-3</td>
<td>63</td>
<td>C</td>
<td>12-3</td>
<td>103</td>
</tr>
<tr>
<td>04</td>
<td>D</td>
<td>12-4</td>
<td>64</td>
<td>D</td>
<td>12-4</td>
<td>104</td>
</tr>
<tr>
<td>05</td>
<td>E</td>
<td>12-5</td>
<td>65</td>
<td>E</td>
<td>12-5</td>
<td>105</td>
</tr>
<tr>
<td>06</td>
<td>F</td>
<td>12-6</td>
<td>66</td>
<td>F</td>
<td>12-6</td>
<td>106</td>
</tr>
<tr>
<td>07</td>
<td>G</td>
<td>12-7</td>
<td>67</td>
<td>G</td>
<td>12-7</td>
<td>107</td>
</tr>
<tr>
<td>08</td>
<td>H</td>
<td>12-8</td>
<td>70</td>
<td>H</td>
<td>12-8</td>
<td>110</td>
</tr>
<tr>
<td>09</td>
<td>I</td>
<td>12-9</td>
<td>71</td>
<td>I</td>
<td>12-9</td>
<td>111</td>
</tr>
<tr>
<td>10</td>
<td>J</td>
<td>11-1</td>
<td>41</td>
<td>J</td>
<td>11-1</td>
<td>112</td>
</tr>
<tr>
<td>11</td>
<td>K</td>
<td>11-2</td>
<td>42</td>
<td>K</td>
<td>11-2</td>
<td>113</td>
</tr>
<tr>
<td>12</td>
<td>L</td>
<td>11-3</td>
<td>43</td>
<td>L</td>
<td>11-3</td>
<td>114</td>
</tr>
<tr>
<td>13</td>
<td>M</td>
<td>11-4</td>
<td>44</td>
<td>M</td>
<td>11-4</td>
<td>115</td>
</tr>
<tr>
<td>14</td>
<td>N</td>
<td>11-5</td>
<td>45</td>
<td>N</td>
<td>11-5</td>
<td>116</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>11-6</td>
<td>46</td>
<td>O</td>
<td>11-6</td>
<td>117</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>11-7</td>
<td>47</td>
<td>P</td>
<td>11-7</td>
<td>118</td>
</tr>
<tr>
<td>17</td>
<td>Q</td>
<td>11-8</td>
<td>50</td>
<td>Q</td>
<td>11-8</td>
<td>121</td>
</tr>
<tr>
<td>18</td>
<td>R</td>
<td>11-9</td>
<td>51</td>
<td>R</td>
<td>11-9</td>
<td>122</td>
</tr>
<tr>
<td>19</td>
<td>S</td>
<td>0-2</td>
<td>22</td>
<td>S</td>
<td>0-2</td>
<td>123</td>
</tr>
<tr>
<td>20</td>
<td>T</td>
<td>0-3</td>
<td>23</td>
<td>T</td>
<td>0-3</td>
<td>124</td>
</tr>
<tr>
<td>21</td>
<td>U</td>
<td>0-4</td>
<td>24</td>
<td>U</td>
<td>0-4</td>
<td>125</td>
</tr>
<tr>
<td>22</td>
<td>V</td>
<td>0-5</td>
<td>25</td>
<td>V</td>
<td>0-5</td>
<td>126</td>
</tr>
<tr>
<td>23</td>
<td>W</td>
<td>0-6</td>
<td>26</td>
<td>W</td>
<td>0-6</td>
<td>127</td>
</tr>
<tr>
<td>24</td>
<td>X</td>
<td>0-7</td>
<td>27</td>
<td>X</td>
<td>0-7</td>
<td>128</td>
</tr>
<tr>
<td>25</td>
<td>Y</td>
<td>0-8</td>
<td>30</td>
<td>Y</td>
<td>0-8</td>
<td>129</td>
</tr>
<tr>
<td>26</td>
<td>Z</td>
<td>0-9</td>
<td>31</td>
<td>Z</td>
<td>0-9</td>
<td>130</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>080</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>081</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>02</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>082</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>03</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>083</td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td>04</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>084</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>05</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>085</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>06</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>086</td>
</tr>
<tr>
<td>34</td>
<td>7</td>
<td>07</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>087</td>
</tr>
<tr>
<td>35</td>
<td>8</td>
<td>08</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>088</td>
</tr>
<tr>
<td>36</td>
<td>9</td>
<td>09</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>089</td>
</tr>
<tr>
<td>37</td>
<td>+</td>
<td>12</td>
<td>60</td>
<td>*</td>
<td>12-8-6</td>
<td>063</td>
</tr>
<tr>
<td>38</td>
<td>-</td>
<td>11</td>
<td>40</td>
<td>-</td>
<td>11-8-4</td>
<td>052</td>
</tr>
<tr>
<td>39</td>
<td>,</td>
<td>11-8-4</td>
<td>54</td>
<td>, (comma)</td>
<td>0-8-3</td>
<td>054</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td>50</td>
<td>, (period)</td>
<td>12-8-3</td>
<td>058</td>
</tr>
<tr>
<td>41</td>
<td>,</td>
<td>12-8-3</td>
<td>73</td>
<td>#</td>
<td>6-3</td>
<td>043</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td>60</td>
<td>;</td>
<td>12-8-2</td>
<td>133</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td>61</td>
<td>1</td>
<td>12-8-2</td>
<td>133</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td>62</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td>63</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td>64</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td>65</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>66</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td>67</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>68</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td>69</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td>70</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td>71</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td>72</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td>73</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
<td>74</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
<td>75</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td>76</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
<td>77</td>
<td>12-8-2</td>
<td>135</td>
<td></td>
</tr>
</tbody>
</table>

† Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.
‡‡ In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55g).
<table>
<thead>
<tr>
<th>Collating Sequence Decimal/Octal</th>
<th>CDC Graphic</th>
<th>Display Code</th>
<th>External BCD</th>
<th>Collating Sequence Decimal/Octal</th>
<th>CDC Graphic</th>
<th>Display Code</th>
<th>External BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>blank</td>
<td>55</td>
<td>20</td>
<td>32 40</td>
<td>H</td>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>01 01</td>
<td>≤</td>
<td>74</td>
<td>15</td>
<td>33 41</td>
<td>I</td>
<td>11</td>
<td>71</td>
</tr>
<tr>
<td>02 02</td>
<td>%</td>
<td>63 †</td>
<td>16 †</td>
<td>34 42</td>
<td>v</td>
<td>66</td>
<td>52</td>
</tr>
<tr>
<td>03 03</td>
<td>(</td>
<td>61</td>
<td>17</td>
<td>35 43</td>
<td>J</td>
<td>12</td>
<td>41</td>
</tr>
<tr>
<td>04 04</td>
<td>→</td>
<td>65</td>
<td>35</td>
<td>36 44</td>
<td>K</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>05 05</td>
<td>=</td>
<td>60</td>
<td>36</td>
<td>37 45</td>
<td>L</td>
<td>14</td>
<td>43</td>
</tr>
<tr>
<td>06 06</td>
<td>^</td>
<td>67</td>
<td>37</td>
<td>38 46</td>
<td>M</td>
<td>15</td>
<td>44</td>
</tr>
<tr>
<td>07 07</td>
<td>↑</td>
<td>70</td>
<td>55</td>
<td>39 47</td>
<td>N</td>
<td>16</td>
<td>45</td>
</tr>
<tr>
<td>08 10</td>
<td>↓</td>
<td>71</td>
<td>56</td>
<td>40 50</td>
<td>O</td>
<td>17</td>
<td>46</td>
</tr>
<tr>
<td>09 11</td>
<td>&gt;</td>
<td>73</td>
<td>57</td>
<td>41 51</td>
<td>P</td>
<td>20</td>
<td>47</td>
</tr>
<tr>
<td>10 12</td>
<td>≥</td>
<td>75</td>
<td>75</td>
<td>42 52</td>
<td>Q</td>
<td>21</td>
<td>50</td>
</tr>
<tr>
<td>11 13</td>
<td>—</td>
<td>76</td>
<td>76</td>
<td>43 53</td>
<td>R</td>
<td>22</td>
<td>51</td>
</tr>
<tr>
<td>12 14</td>
<td>.</td>
<td>57</td>
<td>73</td>
<td>44 54</td>
<td>J</td>
<td>62</td>
<td>32</td>
</tr>
<tr>
<td>13 15</td>
<td>)</td>
<td>52</td>
<td>74</td>
<td>45 55</td>
<td>S</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>14 16</td>
<td>;</td>
<td>77</td>
<td>77</td>
<td>46 56</td>
<td>T</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>15 17</td>
<td>+</td>
<td>45</td>
<td>60</td>
<td>47 57</td>
<td>U</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>16 20</td>
<td>$</td>
<td>53</td>
<td>53</td>
<td>48 60</td>
<td>V</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>17 21</td>
<td>*</td>
<td>47</td>
<td>54</td>
<td>49 61</td>
<td>W</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>18 22</td>
<td>-</td>
<td>46</td>
<td>40</td>
<td>50 62</td>
<td>X</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>19 23</td>
<td>/</td>
<td>50</td>
<td>21</td>
<td>51 63</td>
<td>Y</td>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>20 24</td>
<td>,</td>
<td>56</td>
<td>33</td>
<td>52 64</td>
<td>Z</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>21 25</td>
<td>(</td>
<td>51</td>
<td>34</td>
<td>53 65</td>
<td>:</td>
<td>00 †</td>
<td>none †</td>
</tr>
<tr>
<td>22 26</td>
<td>=</td>
<td>54</td>
<td>13</td>
<td>54 66</td>
<td>0</td>
<td>33</td>
<td>12</td>
</tr>
<tr>
<td>23 27</td>
<td>≠</td>
<td>64</td>
<td>14</td>
<td>55 67</td>
<td>1</td>
<td>34</td>
<td>01</td>
</tr>
<tr>
<td>24 30</td>
<td>&lt;</td>
<td>72</td>
<td>72</td>
<td>56 70</td>
<td>2</td>
<td>35</td>
<td>02</td>
</tr>
<tr>
<td>25 31</td>
<td>A</td>
<td>01</td>
<td>61</td>
<td>57 71</td>
<td>3</td>
<td>36</td>
<td>03</td>
</tr>
<tr>
<td>26 32</td>
<td>B</td>
<td>02</td>
<td>62</td>
<td>58 72</td>
<td>4</td>
<td>37</td>
<td>04</td>
</tr>
<tr>
<td>27 33</td>
<td>C</td>
<td>03</td>
<td>63</td>
<td>59 73</td>
<td>5</td>
<td>40</td>
<td>05</td>
</tr>
<tr>
<td>28 34</td>
<td>D</td>
<td>04</td>
<td>64</td>
<td>60 74</td>
<td>6</td>
<td>41</td>
<td>06</td>
</tr>
<tr>
<td>29 35</td>
<td>E</td>
<td>05</td>
<td>65</td>
<td>61 75</td>
<td>7</td>
<td>42</td>
<td>07</td>
</tr>
<tr>
<td>30 36</td>
<td>F</td>
<td>06</td>
<td>66</td>
<td>62 76</td>
<td>8</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>31 37</td>
<td>G</td>
<td>07</td>
<td>67</td>
<td>63 77</td>
<td>9</td>
<td>44</td>
<td>11</td>
</tr>
</tbody>
</table>

† In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.
<table>
<thead>
<tr>
<th>Collating Sequence Decimal/Octal</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
<th>Collating Sequence Decimal/Octal</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>blank</td>
<td>55</td>
<td>20</td>
<td>32 40</td>
<td>@</td>
<td>74</td>
<td>40</td>
</tr>
<tr>
<td>01 01</td>
<td>!</td>
<td>66</td>
<td>21</td>
<td>33 41</td>
<td>A</td>
<td>01</td>
<td>41</td>
</tr>
<tr>
<td>02 02</td>
<td>&quot;</td>
<td>64</td>
<td>22</td>
<td>34 42</td>
<td>B</td>
<td>02</td>
<td>42</td>
</tr>
<tr>
<td>03 03</td>
<td>#</td>
<td>60</td>
<td>23</td>
<td>35 43</td>
<td>C</td>
<td>03</td>
<td>43</td>
</tr>
<tr>
<td>04 04</td>
<td>$</td>
<td>53</td>
<td>24</td>
<td>36 44</td>
<td>D</td>
<td>04</td>
<td>44</td>
</tr>
<tr>
<td>05 05</td>
<td>%</td>
<td>63†</td>
<td>25</td>
<td>37 45</td>
<td>E</td>
<td>05</td>
<td>45</td>
</tr>
<tr>
<td>06 06</td>
<td>&amp;</td>
<td>67</td>
<td>26</td>
<td>38 46</td>
<td>F</td>
<td>06</td>
<td>46</td>
</tr>
<tr>
<td>07 07</td>
<td>,</td>
<td>70</td>
<td>27</td>
<td>39 47</td>
<td>G</td>
<td>07</td>
<td>47</td>
</tr>
<tr>
<td>08 10</td>
<td>(</td>
<td>51</td>
<td>28</td>
<td>40 50</td>
<td>H</td>
<td>10</td>
<td>48</td>
</tr>
<tr>
<td>09 11</td>
<td>)</td>
<td>52</td>
<td>29</td>
<td>41 51</td>
<td>I</td>
<td>11</td>
<td>49</td>
</tr>
<tr>
<td>10 12</td>
<td>*</td>
<td>47</td>
<td>2A</td>
<td>42 52</td>
<td>J</td>
<td>12</td>
<td>4A</td>
</tr>
<tr>
<td>11 13</td>
<td>+</td>
<td>45</td>
<td>2B</td>
<td>43 53</td>
<td>K</td>
<td>13</td>
<td>4B</td>
</tr>
<tr>
<td>12 14</td>
<td>.</td>
<td>56</td>
<td>2C</td>
<td>44 54</td>
<td>L</td>
<td>14</td>
<td>4C</td>
</tr>
<tr>
<td>13 15</td>
<td>-</td>
<td>46</td>
<td>2D</td>
<td>45 55</td>
<td>M</td>
<td>15</td>
<td>4D</td>
</tr>
<tr>
<td>14 16</td>
<td>.</td>
<td>57</td>
<td>2E</td>
<td>46 56</td>
<td>N</td>
<td>16</td>
<td>4E</td>
</tr>
<tr>
<td>15 17</td>
<td>/</td>
<td>50</td>
<td>2F</td>
<td>47 57</td>
<td>O</td>
<td>17</td>
<td>4F</td>
</tr>
<tr>
<td>16 20</td>
<td>0</td>
<td>33</td>
<td>30</td>
<td>48 60</td>
<td>P</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>17 21</td>
<td>1</td>
<td>34</td>
<td>31</td>
<td>49 61</td>
<td>Q</td>
<td>21</td>
<td>51</td>
</tr>
<tr>
<td>18 22</td>
<td>2</td>
<td>35</td>
<td>32</td>
<td>50 62</td>
<td>R</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>19 23</td>
<td>3</td>
<td>36</td>
<td>33</td>
<td>51 63</td>
<td>S</td>
<td>23</td>
<td>53</td>
</tr>
<tr>
<td>20 24</td>
<td>4</td>
<td>37</td>
<td>34</td>
<td>52 64</td>
<td>T</td>
<td>24</td>
<td>54</td>
</tr>
<tr>
<td>21 25</td>
<td>5</td>
<td>40</td>
<td>35</td>
<td>53 65</td>
<td>U</td>
<td>25</td>
<td>55</td>
</tr>
<tr>
<td>22 26</td>
<td>6</td>
<td>41</td>
<td>36</td>
<td>54 66</td>
<td>V</td>
<td>26</td>
<td>56</td>
</tr>
<tr>
<td>23 27</td>
<td>7</td>
<td>42</td>
<td>37</td>
<td>55 67</td>
<td>W</td>
<td>27</td>
<td>57</td>
</tr>
<tr>
<td>24 30</td>
<td>8</td>
<td>43</td>
<td>38</td>
<td>56 70</td>
<td>X</td>
<td>30</td>
<td>58</td>
</tr>
<tr>
<td>25 31</td>
<td>9</td>
<td>44</td>
<td>39</td>
<td>57 71</td>
<td>Y</td>
<td>31</td>
<td>59</td>
</tr>
<tr>
<td>26 32</td>
<td>:)</td>
<td>00†</td>
<td>3A</td>
<td>58 72</td>
<td>Z</td>
<td>32</td>
<td>5A</td>
</tr>
<tr>
<td>27 33</td>
<td>;</td>
<td>77</td>
<td>3B</td>
<td>59 73</td>
<td>[</td>
<td>61</td>
<td>5B</td>
</tr>
<tr>
<td>28 34</td>
<td>&lt;</td>
<td>72</td>
<td>3C</td>
<td>60 74</td>
<td>\</td>
<td>75</td>
<td>5C</td>
</tr>
<tr>
<td>29 35</td>
<td>=</td>
<td>54</td>
<td>3D</td>
<td>61 75</td>
<td>\</td>
<td>62</td>
<td>5D</td>
</tr>
<tr>
<td>30 36</td>
<td>&gt;</td>
<td>73</td>
<td>3E</td>
<td>62 76</td>
<td>^</td>
<td>76</td>
<td>5E</td>
</tr>
<tr>
<td>31 37</td>
<td>?</td>
<td>71</td>
<td>3F</td>
<td>63 77</td>
<td>_</td>
<td>65</td>
<td>5F</td>
</tr>
</tbody>
</table>

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.
### GLOSSARY

**Access Control** -
Protection of data from unauthorized access or modification.

**Actual Key** -
A file organization in which records are stored according to their system-assigned key values.

**Advanced Access Methods (AAM)** -
A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor. See CYBER Record Manager.

**Alias** -
A data name used in the sub-schema in place of a schema data name.

**Area** -
A uniquely named schema data base subdivision that contains data records; identified in the sub-schema as a realm; a file in the operating system.

**Automatic Recovery** -
CDCS initiated recovery operations that make a data base usable and consistent after some type of software or hardware failure (but not media failure).

**Basic Access Methods (BAM)** -
A file manager that processes sequential and word addressable file organizations. See CYBER Record Manager.

**CDCS** -
See CYBER Database Control System.

**CDCS Batch Test Facility** -
A facility that allows an application to simulate a data base environment without impacting any other CDCS users on the system.

**Child Record Occurrence** -
For relation processing, a record occurrence that has another record occurrence (the parent record occurrence) at the next numerically lower rank in the relation.

**Common Item** -
A data item that appears in two or more files joined in a relation; in each instance, the data item contains the same value.

**Concurrency** -
Simultaneous access to the same data in a data base by two or more application programs during a given span of time.

**Constraint** -
A control imposed on records in related files or on items in a single file for the purpose of protecting the integrity of data in a data base during update operations. A constraint is defined in the schema and is based on the common item in the records.

**Control Break** -
A condition that occurs during a relation read to signify a new record occurrence was read for the parent file.

**CRM** -
See CYBER Record Manager.

**CYBER Database Control System (CDCS)** -
The controlling module that provides the interface between the application program and the data base.

**CYBER Record Manager (CRM)** -
A generic term relating to the common products BAM and AAM, which run under the NOS and NOS/BE operating systems and allow a variety of record types, block types, and file organizations to be created and accessed. The execution time input/output of the DMS-170 products is implemented through CRM. All CRM file processing requests ultimately pass through the operating system input/output routines.

**Data Administrator** -
A person who defines the format and organization of the data base.

**Data Base** -
A systematically organized, central pool of information; organization is described by a schema.

**Data Base Procedure** -
A special-purpose routine that performs a predefined operation; specified in the schema and initiated by CDCS.

**Data Base Status Block** -
An array defined within an application program to which CDCS returns information concerning the status of operations on data base files and relations. The status block is updated after each CDCS operation.

**Data Base Transaction** -
A series of update operations identified by a user-assigned transaction identifier. A data base transaction is bracketed by a begin transaction operation and either a commit or drop operation. Data base transactions also provide a program restart capability that can be used for restarting an application program after a system failure.

**Data Base Version** -
A set of data files that is described by a schema. Data base versions are defined in the master directory. When data base versions are used, a schema (the description of the data base) can be used with more than one set of files (each set of files being a data base version).

**Data Description Language (DDL)** -
The language used to structure the schema and the sub-schema.
Data Item -
A unit of data within a record; can be a variable or an array in the FORTRAN sub-schema.

Data Manipulation Language (DML) -
The extensions to FORTRAN that provide access to a DMS-170 data base.

DDL -
See Data Description Language.

Deadlock -
A situation that arises in concurrent data base access when two or more application programs, each with locked resources, are contending for a resource that is locked by one of the other application programs, and none of the programs can proceed without that resource.

Dependent Record Occurrence -
A record occurrence that is the dependent member of a condition defined by a constraint.

Direct Access -
In the context of CRM, one of the five file organizations. The organization is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks called home blocks of the file.

In the context of NOS permanent files, a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

DML -
See Data Manipulation Language.

Dominant Record Occurrence -
A record occurrence that is the dominant member of a condition defined by a constraint.

Exclusive Locking -
Locking mechanism that allows one program access to a realm or record and prohibits all access by other users. Contrast with Protected Locking.

File -
A collection of records treated as a unit; an area in the schema; a realm in the sub-schema.

Hierarchical Tree Structure -
A representation that commonly illustrates record occurrences for files joined in a directed relation. The root of the tree is a record occurrence in the root file, and each successive level represents the record occurrences in each joined file.

Home Block -
Mass storage allocated for a file with direct access organization at the time the file is created.

Indexed Sequential -
A file organization in which records are stored in ascending order by key.

Keyword -
A word that is required in a source program statement.

Log Files -
Files that hold historical records of operations performed by users on data base areas.

Mapping -
The process by which CDCS produces a record or item image conforming to the schema or sub-schema description.

Master Directory -
A file created by the data administrator and used by CDCS in processing. This information consists of schema and sub-schema tables, media parameters, and data base procedure library and logging specifications.

Multiple-Index Processor -
A processor that allows AAM files to be accessed by alternate keys.

Null Record Occurrence -
A record occurrence composed of the display code right bracket symbol in each character position. The null record occurrence is used in a relation occurrence to denote that no record occurrence qualifies or that a record occurrence does not exist at a given level in the relation.

Operation -
A particular function performed on units of data; for instance, opening or closing an area, or storing or deleting a record.

Parent Record Occurrence -
For relation processing, a record occurrence that has another record occurrence at the next numerically higher rank in the relation.

Permanent File -
A file that resides on a mass storage permanent file device and can be retained for longer than a single job. The file is protected against accidental destruction and can be protected against unauthorized access.

Privacy Key -
A character constant, variable name, or unsubscripted name that is included in a FORTRAN DML PRIVACY statement to gain access to a particular realm.

Protected Locking -
Locking mechanism that allows one program access to a realm or record for update operations and prohibits update operations (allows read operations) by other users. Contrast with Exclusive Locking.

Rank -
The rank of a file in a relation corresponds to the position of the file in the schema definition of the relation. The ranks of the files joined in a relation are numbered consecutively, with the root file having a rank of 1.

Realm -
A uniquely named sub-schema data base subdivision that contains data records; identified in the schema as an area; a file.
Realm Ordinal -
A unique identifier assigned to each realm in a sub-schema when the sub-schema is compiled. Sub-schema realm ordinals are used in conjunction with the data base status block.

Record -
A named collection of one or more data items that are treated as a unit.

Record Occurrence -
An actual data base record that conforms to a record description in the schema.

Record Type -
The description of the attributes of a record; record layout.

Recovery -
A process that makes a data base useful after some type of software or hardware failure has occurred.

Relation -
A group of files that are related by common data items; therefore the files can be opened, closed, or read by a single request. Relations are defined in the schema.

Relation Occurrence -
The logical concatenation of a record occurrence from each record type specified in the relation.

Restart Identifier -
A unique identifier for a run-unit that is maintained by CDCS for program restart operations in data base transactions.

Restart Identifier File -
A random permanent file used internally by CDCS to support program restart operations for programs that request a restart identifier.

Restriction -
Criteria that must be satisfied by a record occurrence in a relation before it can be made available to the application program. Restrictions are defined in the sub-schema.

Root Realm -
The first realm listed in a relation; the root realm has the rank of 1 in a relation; record occurrences of the root realm are pictured as the root of a tree in a hierarchical tree structure.

Schema -
A detailed description of the internal structure of the complete data base.

Status Block -
See Data Base Status Block.

Sub-Schema -
A detailed description of the portion of the data base to be made available to one or more application programs.

Sub-Schema Item Ordinal -
An identifier, unique within a record, assigned to each item in a sub-schema when the sub-schema is compiled. Sub-schema item ordinals are used in conjunction with the data base status block.

Sub-Schema Library -
A permanent file containing one or more sub-schemas.

Transaction -
See Data Base Transaction.

Transaction Identifier -
A user-assigned identifier for a data base transaction. The identifier is used for program restart operations in data base transactions.
This appendix contains the source programs and control statements used to generate the data base environment for the university application presented in this user's guide. Although all programs reflect operation under the NOS operating system, conversion to the NOS/BE operating system could be accomplished by making the following changes:

1. Substitute the NOS/BE REQUEST and CATALOG control statements for the NOS DEFINE control statement.

2. Substitute the NOS/BE file identification parameter ID for the NOS file identification parameter UN. This substitution applies to the source input for the master directory.

Setting up a DMS-170 data management environment is a data administrator responsibility; the process is shown here, however, to allow the reader to duplicate the application and use it to gain an understanding of DMS-170 FORTRAN application programming. The source input for the jobs shown in this appendix illustrates the university application being created by a series of batch jobs. The source input for each job is shown exactly as required for processing on NOS with two exceptions:

- End-of-record is indicated by the statement end-of-record and a blank line that is inserted to improve readability of the text.

- End of input for the job is indicated by the statement end-of-information.

The steps the data administrator takes to establish the application are listed in appropriate order as follows:

1. Design, write, compile, and store the schema definition as a permanent file library. A FORTRAN sub-schema library is stored as a permanent file named SSLIB. Input consists of four separate sub-schemas (AVERAGE, RELATION, ADMISSIONS, BURSAR); the sub-schemas follow each other with no intervening end-of-records. See figure C-2 for both source input and the listing that results from compilation. The FORTRAN sub-schema CREATES is used to create the data base and is stored in a permanent file library named CREATES. See figure C-3.

2. Generate a master directory through the DBMSTRD utility. A master directory is stored as a permanent file named MSTRDIR. See figure C-4.

3. Initialize log or recovery files specified for the schema in the master directory. The master directory specifies a transaction recovery file and a restart identifier file for schema UNIVERSITY. The DBREC utility is used to initialize these files. For processing on NOS/BE, the series of control statements that must be executed for each log and recovery file before the DBREC control statement is executed is as follows: REQUEST,REWIND,CATALOG,AND RETURN. See figure C-5.

4. Write a program to store the data base. A FORTRAN program creates five data base files (PROFESSOR, COURSE, STUDENT, CURRICULUM, ACCOUNTING) using the FORTRAN sub-schema CREATES, and defines the appropriate index files assigned in the master directory. See figure C-6. CDCSBTF is used to run this program; therefore, CDCS does not have to be active. For processing on NOS/BE, the series of control statements that must be executed for each area and index file before the FTN5 control statement is executed is as follows: REQUEST,REWIND,CATALOG,AND RETURN.

5. Establish CDCS as an active system. This must be done by the data administrator; the process is not shown in this guide. If CDCS is not established as an active system CDCSBTF can be used.
Figure C-1. The UNIVERSITY Schema
SUBSCHEMA AVERAGE, SCHEMA=UNIVERSITY

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM) STUDENT=STUDENT-ID.CURR-REC
ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC

REALM CFILE

RECORD CRECORD

CHARACTER*14 IDENT
CHARACTER*11 STUDENT
CHARACTER*6 COURSE
REAL GRADE
END

SUBSCHEMA COMPARE, SCHEMA=UNIVERSITY

ALIAS(REALM) PFILE=PROFESSOR
ALIAS(RECORD) PRECORD=PROF-REC
ALIAS(ITEM) PROFID=PROF-ID.PROF-REC
ALIAS(ITEM) PNAME=PROF-NAME

ALIAS(REALM) CRSFILE=COURSE
ALIAS(RECORD) CRSREC=COURSE-REC
ALIAS(ITEM) CRSID=COURSE-ID.COURSE-REC
ALIAS(ITEM) CRSNAME=COURSE-NAME
ALIAS(ITEM) PROF=PROF-ID.COURSE-REC
ALIAS(ITEM) FIELD=ACADEMIC-FIELD

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC
ALIAS(ITEM) CODE=COMPLETE-CODE
ALIAS(ITEM) DATE=COMPLETE-DATE

REALM PFILE
REALM CRSFILE
REALM CFILE

RECORD PRECORD
CHARACTER*8 PROFID
CHARACTER*30 PNAME
CHARACTER*20 FIELD

RECORD CRSREC
CHARACTER*6 CRSID
CHARACTER*20 CRSNAME
CHARACTER*8 PROF

RECORD CRECORD
CHARACTER*14 IDENT
CHARACTER*6 COURSE
CHARACTER*1 CODE
CHARACTER*8 DATE
REAL GRADE

RELATION REL3
RESTRICT CRECORD (CODE .EQ. 'C')
END

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 1 of 8)
SUBSCHEMA RELATION, SCHEMA=UNIVERSITY

ALIAS(REALM) SFILE=STUDENT
ALIAS(RECORD) SRECORD=STUDENT-REC
ALIAS(ITEM) STID=STUDENT-ID.STUDENT-REC

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CRECORD=CURR-REC
ALIAS(ITEM) CSTID=STUDENT-ID.CURR-REC
ALIAS(ITEM) COURS=COURSE-ID.CURR-REC
ALIAS(ITEM) PROF=PROF-NAME
ALIAS(ITEM) CODE=COMPLETE-CODE
ALIAS(ITEM) DATE=COMPLETE-DATE

REALM SFILE
REALM CFILE

RECORD SRECORD
CHARACTER*11 STID
CHARACTER*20 MAJOR

RECORD CRECORD
CHARACTER*14 IDENT
CHARACTER*11 CSTID
CHARACTER*6 COURS
REAL GRADE
CHARACTER CODE
CHARACTER*8 DATE
INTEGER UNITS
RELATION REL1
RESTRICT CRECORD(CODE.EQ.'C')
END

SUBSCHEMA ADMISSIONS, SCHEMA=UNIVERSITY

ALIAS(RECORD) CRSREC=COURSE-REC
ALIAS(ITEM) CID=COURSE-ID.COURSE-REC
ALIAS(ITEM) NAME=COURSE-NAME
ALIAS(ITEM) PREREQ=PREREQUISITE

ALIAS(REALM) CFILE=CURRICULUM
ALIAS(RECORD) CURREC=CURR-REC
ALIAS(ITEM) STUDENT=STUDENT-ID
ALIAS(ITEM) CCID=COURSE-ID.CURR-REC
ALIAS(ITEM) CODE=COMPLETE-CODE

REALM COURSE
REALM CFILE

RECORD CRSREC
CHARACTER*6 CID
CHARACTER*20 NAME
CHARACTER*20 SCHOOL
CHARACTER*6 PREREQ
INTEGER UNITS

RECORD CURREC
CHARACTER*14 IDENT
CHARACTER*11 STUDENT
CHARACTER*6 CCID
CHARACTER CODE
END

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 2 of 8)
SUBSCHEMA BURSAR, SCHEMA=UNIVERSITY

ALIAS(RECORD) STREC=STUDENT-REC
ALIAS(ITEM) STID=STUDENT-ID.STUDENT-REC
ALIAS(ITEM) NAME=STUDENT-NAME
ALIAS(ITEM) ADDR=STREET-ADDRESS
ALIAS(ITEM) ZIP=ZIP-CODE

ALIAS(REALM) ACCOUNT=ACCOUNTING
ALIAS(RECORD) ACCTREC=ACCT-REC
ALIAS(ITEM) ASTID=STUDENT-ID.ACCT-REC
ALIAS(ITEM) LAB=LAB-FEES
ALIAS(ITEM) MISC=MISC-FEES

REALM STUDENT
REALM ACCOUNT

RECORD STREC
CHARACTER*11 STID
CHARACTER*30 NAME
CHARACTER*20 ADDR
CHARACTER*10 CITY
CHARACTER*2 STATE
CHARACTER*5 ZIP

RECORD ACCTREC
CHARACTER*11 ASTID
REAL TUITION(16)
REAL LAB(16)
REAL BOOKS(16)
REAL MISC(16)

RELATION REL2
END
End-of-record
End-of-information

Compilation Source Listings

* SOURCE LISTING * (80351) DDLF 1.2+538.

SUBSCHEMA AVERAGE, SCHEMA=UNIVERSITY

00001
00002
00003
00004
00005
00006
00007
00008
00009
00010
00011

** WITHIN CFILE
** ORDINAL 1
** ORDINAL 2
** ORDINAL 3
** ORDINAL 4

00012
00013
00014
00015
00016

CHARACTER*14 IDENT
CHARACTER*11 STUDENT
CHARACTER*6 COURSE
REAL GRADE
END

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 3 of 8)
END OF SUB-SCHMA SOURCE INPUT

PRIMARY KEY 00012
ALTERNATE KEY 00013
ALTERNATE KEY 00014
ALTERNATE KEY 00015

****

RECORD MAPPING IS NEEDED FOR REALM - CFILE

BEGIN SUB-SCHMA FILE MAINTENANCE

SUBSCHEMA AVERAGE CHECKSUM

AVERAGE

DDLF COMPLETE.

0 DIAGNOSTICS.

47600B CM USED.

0.064 CP SECS.

COMPARE

* SOURCE LISTING * (80351) DDLF 1.2+538.

00001 SUBSCHEMA COMPARE,SCHEMA=UNIVERSITY
00002
00003 ALIAS(REALM) PFILE=PROFESSOR
00004 ALIAS(RECORD) PRECORD=PROF-REC
00005 ALIAS(ITEM) PROFID=PROF-ID.PROF-REC
00006 ALIAS(ITEM) PNAME=PROF-NAME
00007
00008 ALIAS(REALM) CRSFILE=COURSE
00009 ALIAS(RECORD) CRSREC=COURSE-REC
00010 ALIAS(ITEM) CRSID=COURSE-ID.COURSE-REC
00011 ALIAS(ITEM) CRSNAME=COURSE-NAME
00012 ALIAS(ITEM) PROF=PROF-ID.COURSE-REC
00013 ALIAS(ITEM) FIELD=ACADEMIC-FIELD
00014
00015 ALIAS(REALM) CFILE=CURRICULUM
00016 ALIAS(RECORD) CRECORD=CURR-REC
00017 ALIAS(ITEM) COURSE=COURSE-ID.CURR-REC
00018 ALIAS(ITEM) CODE=COMPLETE-CODE
00019 ALIAS(ITEM) DATE=COMPLETE-DATE
00020
00021 REALM PFILE
00022 REALM CRSFILE
00023 REALM CFILE
00024
00025 ** WITHIN PFILE
00026 00026 CHARACTER=8 PROFID
00027
00028 CHARACTER=20 FIELD
00029
00030 ** WITHIN CRSFILE
00031 00031 CHARACTER=6 CRSID
00032
00033 CHARACTER=20 CRSSNAME
00034
00035 ** ORDINAL 3
00036
00037 ** ORDINAL 3

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 4 of 8)
** WITHIN CFILE
** ORDINAL 1
** ORDINAL 2
** ORDINAL 3
** ORDINAL 4
** ORDINAL 5

PRIMARY KEY 00040
ALTERNATE KEY 00041

RECORD CRECORD
CHARACTER*14 IDENT
CHARACTER*6 COURSE
CHARACTER*1 CODE
CHARACTER*8 DATE
REAL GRADE

RELATION REL3
PROFID FOR AREA PFILE
FIELD FOR AREA PFILE
CRSID FOR AREA CRSFILE
PROF FOR AREA CRSFILE
IDENT FOR AREA CFILE
COURSE FOR AREA CFILE
GRADE FOR AREA CFILE

***** RECORD MAPPING IS NOT NEEDED FOR REALM - PFILE
***** RECORD MAPPING IS NEEDED FOR REALM - CRSFILE
***** RECORD MAPPING IS NEEDED FOR REALM - CFILE

RESTRICT CRECORD (CODE .EQ. 'C')
END

END OF SUB-SCHEMA SOURCE INPUT

RELATION 001
REL3 JOINS
AREA - PFILE
AREA - CRSFILE
AREA - CFILE

BEGIN SUB-SCHEMA FILE MAINTENANCE

SUBSCHEMA COMPARE
CHECKSUM
71111404530456653576

END OF FILE MAINTENANCE

DIAGNOSTICS.
0.146 CP SECS.

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 5 of 8)
** SOURCE LISTING ** (80351) DDLF 1.2+538.

RELATION

| 00001 | SUBSCHEMA RELATION, SCHEMA=UNIVERSITY |
| 00002 |
| 00003 | ALIAS(REALM) SFILE=STUDENT |
| 00004 | ALIAS(RECORD) SRECORD=STUDENT-REC |
| 00005 | ALIAS(ITEM) STID=STUDENT-ID.STUDENT-REC |
| 00006 |
| 00007 | ALIAS(REALM) CFILE=CURRICULUM |
| 00008 | ALIAS(RECORD) CRECORD=CURR-REC |
| 00009 | ALIAS(ITEM) CSTID=STUDENT-ID.CURR-REC |
| 00010 | ALIAS(ITEM) COURS=COURSE-ID.CURR-REC |
| 00011 | ALIAS(ITEM) PROF=PROF-NAME |
| 00012 | ALIAS(ITEM) CODE=COMPLETE-CODE |
| 00013 | ALIAS(ITEM) DATE=COMPLETE-DATE |
| 00014 |
| 00015 | REALM SFILE |
| 00016 | REALM CFILE |
| 00017 |
| 00018 | RECORD SRECORD |
| ** WITHIN SFILE ** |
| 00019 | CHARACTER=11 STID |
| 00020 | CHARACTER=20 MAJOR |
| 00021 |
| ** ORDINAL 2 ** |
| 00022 | RECORD CRECORD |
| ** WITHIN CFILE ** |
| 00023 | CHARACTER=14 IDENT |
| 00024 | CHARACTER=11 CSTID |
| 00025 | CHARACTER=6 COURS |
| 00026 | REAL GRADE |
| 00027 | CHARACTER CODE |
| ** ORDINAL 5 ** |
| 00028 | CHARACTER=8 DATE |
| 00029 | INTEGER UNITS |
| ** ORDINAL 6 ** |
| 00030 | RELATION REL1 |
| PRIMARY KEY 00019 | STID FOR AREA SFILE |
| ALTERNATE KEY 00020 | MAJOR FOR AREA SFILE |
| PRIMARY KEY 00023 | IDENT FOR AREA CFILE |
| ALTERNATE KEY 00024 | CSTID FOR AREA CFILE |
| ALTERNATE KEY 00025 | COURS FOR AREA CFILE |
| ALTERNATE KEY 00026 | GRADE FOR AREA CFILE |
| ***** | RECORD MAPPING IS NEEDED FOR REALM - SFILE |
| ***** | RECORD MAPPING IS NEEDED FOR REALM - CFILE |
| 00031 | RESTRICT CRECORD(CODE.EQ.'C') |
| 00032 | END |
| 00033 | END OF SUB-SCHEMA SOURCE INPUT |

******* RELATION STATISTICS *******

RELATION 001

REL1 JoINS

AREA - SFILE

AREA - CFILE

--- BEGIN SUB-SCHEMA FILE MAINTENANCE ---

SUBSCHEMA

RELATION

CHECKSUM

76710464332261536703

--- END OF FILE MAINTENANCE ---

DDLFL COMPLETE.

0 DIAGNOSTICS.

50500B CM USED.

0.128 CP SECS.

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 6 of 8)
ADMISSIONS

00001 SUBSCHEMA ADMISSIONS,SCHEMA=UNIVERSITY
00002
00003 ALIAS(RECORD) CRSREC=COURSE-REC
00004 ALIAS(ITEM) CID=COURSE-ID.COURSE-REC
00005 ALIAS(ITEM) NAME=COURSE-NAME
00006 ALIAS(ITEM) PREREQ=PREREQUISITE
00007
00008 ALIAS(REALM) CFILE=CURRICULUM
00009 ALIAS(RECORD) CURREC=CURR-REC
00010 ALIAS(ITEM) STUDENT=STUDENT-ID
00011 ALIAS(ITEM) CCID=COURSE-ID.CURR-REC
00012 ALIAS(ITEM) CODE=COMPLETE-CODE
00013
00014 REALM COURSE
00015 REALM CFILE
00016
00017 RECORD CRSREC
00018 ** WITHIN COURSE
00019 ** ORDINAL 1
00020 00018 CHARACTER*6 CID
00021 ** ORDINAL 2
00022 00019 CHARACTER*20 NAME
00023 ** ORDINAL 3
00024 00020 CHARACTER*20 SCHOOL
00025 ** ORDINAL 4
00026 00021 CHARACTER*6 PREREQ
00027 ** ORDINAL 5
00028 00022 INTEGER UNITS
00029 00023
00030 ** WITHIN CFILE
00031 ** ORDINAL 1
00032 00024 RECORD CURREC
00033 ** ORDINAL 2
00034 00025 CHARACTER*14 IDENT
00035 ** ORDINAL 3
00036 00026 CHARACTER*11 STUDENT
00037 ** ORDINAL 4
00038 00027 CHARACTER*6 CCID
00039 00028 CHARACTER CODE
00040 00029 END
00041 00030
00042 ***** END OF SUB-SCHEMA SOURCE INPUT
00043
00044 PRIMARY KEY 00018 CID FOR AREA COURSE
00045 PRIMARY KEY 00025 IDENT FOR AREA CFILE
00046 ALTERNATE KEY 00026 STUDENT FOR AREA CFILE
00047 ALTERNATE KEY 00027 CCID FOR AREA CFILE
00048 ***** RECORD MAPPING IS NEEDED FOR REALM - COURSE
00049 ***** RECORD MAPPING IS NEEDED FOR REALM - CFILE
00050
00051 ---- BEGIN SUB-SCHEMA FILE MAINTENANCE ----

00052 SUBSCHEMA ADMISSIONS
00053 CHECKSUM 56065313377542307610

00054 DDLF COMPLETE. 0 DIAGNOSTICS.
00055 50000B CM USED. 0.109 CP SECS.

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 7 of 8)
SUBSCHEMA BURSAR, SCHEMA=UNIVERSITY
ALIAS(RECORD) STREC=STUDENT-REC
ALIAS(ITEM) STID=STUDENT-ID, STUDENT-REC
ALIAS(ITEM) NAME=STUDENT-NAME
ALIAS(ITEM) ADDR=STREET-ADDRESS
ALIAS(ITEM) ZIP=ZIP-CODE
ALIAS(REALM) ACCOUNT=ACCOUNTING
ALIAS(RECORD) ACCTREC=ACCT-REC
ALIAS(ITEM) ASTID=STUDENT-ID, ACCT-REC
ALIAS(ITEM) LAB=LAB-FEES
ALIAS(ITEM) MISC=MISC-FEES
REALM STUDENT
REALM ACCOUNT
** WITHIN STUDENT
** ORDINAL 1
** ORDINAL 2
** ORDINAL 3
** ORDINAL 4
** ORDINAL 5
** ORDINAL 6
** WITHIN ACCOUNT
** ORDINAL 1
** ORDINAL 2
** ORDINAL 3
** ORDINAL 4
** ORDINAL 5

PRIMARY KEY 00019
PRIMARY KEY 00027

RELA TION REL2

RELATION 001
REL2 JOINS AREA - STUDENT
AREA - ACCOUNT

BEGIN SUB-SCHEMA FILE MAINTENANCE

CHECKSUM
1664353141007716046

DDLF COMPLETE.
0 DIAGNOSTICS.
505008 CM USED.
0.107 CP SECS.

Figure C-2. The FORTRAN Sub-Schema Library (Sheet 8 of 8)
Figure C-3. The FORTRAN Sub-Schema Library CREATES
AREA NAME IS CURRICULUM
PFN IS "CURRUCU" UN IS "DBID001"
INDEX FILE ASSIGNED
PFN "CRNDX" UN IS "DBID001".

AREA NAME IS ACCOUNTING
PFN IS "ACCOUNT" UN IS "DBID001".
SUBSCHEMA NAME IS AVERAGE
FILE NAME IS SSLIB.

SUBSCHEMA NAME IS COMPARE
FILE NAME IS SSLIB.
SUBSCHEMA NAME IS RELATION
FILE NAME IS SSLIB.
SUBSCHEMA NAME IS ADMISSIONS
FILE NAME IS CREATES.
SUBSCHEMA NAME IS CREATES
FILE NAME IS CREATES.
SUBSCHEMA NAME IS BURSAR
FILE NAME IS SSLIB.
SUBSCHEMA NAME IS CREATES
FILE NAME IS CREATES.
SUBSCHEMA NAME IS CREATES
FILE NAME IS CREATES.

End-of-Record
End-of-Information

Figure C-5. Log File Initialization
Figure C-6. The FORTRAN Data Base Creation Program (Sheet 1 of 3)
File PDATA—(Input data for PFILE)

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRLN0080</td>
<td>CARLIN, W.L.</td>
<td>HISTORY</td>
</tr>
<tr>
<td>DVS00575D</td>
<td>DAVIDS, M.E.</td>
<td>PSYCHOLOGY</td>
</tr>
<tr>
<td>JCKSN750J</td>
<td>JACKSON, U.B.</td>
<td>BUSINESS</td>
</tr>
<tr>
<td>JMS00160J</td>
<td>JAMES, H.L.</td>
<td>PSYCHOLOGY</td>
</tr>
<tr>
<td>MLN0084G</td>
<td>MALONE, R.E.</td>
<td>HISTORY</td>
</tr>
<tr>
<td>RSS00860R</td>
<td>ROSS, W.R.</td>
<td>BUSINESS</td>
</tr>
<tr>
<td>WLSN0855W</td>
<td>WILSON, G.R.</td>
<td>CHEMISTRY</td>
</tr>
<tr>
<td>YMD00170Y</td>
<td>YAMADA, J.V.</td>
<td>BUSINESS</td>
</tr>
</tbody>
</table>

File CRSDATA—(Input data for CRSFILE)

<table>
<thead>
<tr>
<th>Code</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHM0103</td>
<td>BIOCHEMISTRY</td>
</tr>
<tr>
<td>CHM005Q</td>
<td>QUANTITATIVE ANALYSIS</td>
</tr>
<tr>
<td>CHM101L</td>
<td>LINEAR OPTIMIZATION</td>
</tr>
<tr>
<td>PSY136G</td>
<td>SOCIAL PSYCHOLOGY</td>
</tr>
<tr>
<td>PSY002G</td>
<td>GENERAL PSYCHOLOGY</td>
</tr>
<tr>
<td>HIS103G</td>
<td>GREEK HISTORY</td>
</tr>
<tr>
<td>BUS001A</td>
<td>BUSINESS ADMIN</td>
</tr>
<tr>
<td>BUS002A</td>
<td>BUSINESS ADMIN</td>
</tr>
<tr>
<td>MATH10C</td>
<td>COLLEGE ALGEBRA</td>
</tr>
</tbody>
</table>

File SDATA—(Input data for STUDENT)

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>122-13-6704W</td>
<td>WALTER HILL</td>
<td>1960 MONTANA ST.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>100-22-5860G</td>
<td>GUY RICHARDS</td>
<td>143 E. LAKE BLVD.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-7159</td>
<td>124-33-5780</td>
<td>413 MAPLE AVE.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-7315</td>
<td>JERI ADAMS</td>
<td>1400 W. OAK LANES</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-6252</td>
<td>120-44-3760</td>
<td>137 MARKET ST.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>553-89-2021P</td>
<td>PAUL JOHNSON</td>
<td>100 KAUREL DR.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-6490</td>
<td>197-11-2140</td>
<td>102 MORRIS ST.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-1369</td>
<td>678-12-1144</td>
<td>1372 PARKVIEW DR.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>387-14-1232L</td>
<td>LLOYD DAVIS</td>
<td>692 FIRST ST. APT.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-9934</td>
<td>437-56-8943</td>
<td>986 SINCLAIRE AVE.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
<tr>
<td>MNN5512612-9761</td>
<td>553-89-2021</td>
<td>127 MARKET ST.</td>
<td>MINNEAPOLI</td>
<td></td>
</tr>
</tbody>
</table>

Figure C-6. The FORTRAN Data Base Creation Program (Sheet 2 of 3)
File CDATA—(Input data for CFILE)

122-13-6704-01122-13-6704HIS1034.0C09/22/803.0
122-13-6704-02122-13-6704PSY1360.0I 3.0
100-22-5860-01100-22-5860CHI1033.0C05/30/793.0
100-22-5860-02100-22-5860CHM0054.0C09/22/794.0
100-22-5860-03100-22-5860MATH1035.0C05/18/793.0
100-22-5860-04100-22-5860PSY0020.0I 3.0
124-33-5780-01124-33-5780HIS1033.5C09/22/803.0
124-33-5780-02124-33-5780BUS0024.0C02/24/803.0
120-44-3760-01120-44-3760CHM1033.0C05/30/803.0
120-44-3760-02120-44-3760MATH104.0C05/30/803.0
120-44-3760-03120-44-3760MATH104.0C05/30/803.0
120-44-3760-04120-44-3760CHM1103.5C09/22/803.0
553-89-2021-01553-89-2021PSY1363.0C05/30/803.0
553-89-2021-02553-89-2021PSY0024.0C05/30/803.0
553-89-2021-03553-89-2021PSY0033.0C09/22/803.0
687-14-2100-01687-14-2100CHM0054.0C09/22/794.0
687-14-2100-02687-14-2100CHM0054.0C05/30/803.0
687-14-2100-03687-14-2100CHM0054.0C05/30/803.0
687-14-2100-04687-14-2100CHM0054.0C05/30/803.0
197-11-2140-01197-11-2140CHM0053.0C05/30/804.0
197-11-2140-02197-11-2140CHM0053.0C09/22/803.0
197-11-2140-03197-11-2140CHM0053.0C09/22/803.0
197-11-2140-04197-11-2140CHM0053.0C09/22/803.0
678-12-1144-01678-12-1144PSY1364.0C05/30/803.0
678-12-1144-02678-12-1144BUS0174.0C05/30/803.0
678-12-1144-03678-12-1144HIS1030.0I 3.0
387-14-1232-01387-14-1232MATH1000.0I 3.0
387-14-1232-02387-14-1232CHM0050.0I 4.0
387-14-1232-03387-14-1232CHM0050.0I 3.0
387-14-1232-04387-14-1232BUS0170.0I 3.0
437-56-8943-01437-56-8943MATH1035.0C05/30/803.0
437-56-8943-02437-56-8943CHM0053.0C05/30/804.0
437-56-8943-03437-56-8943PSY0023.0C05/30/803.0

File ACCDATA—(Input data for ACCOUNT)

122-13-6704
100-22-5860
124-33-5780
120-44-3760
553-89-2021
687-14-2100
197-11-2140
678-12-1144
387-14-1232
437-56-8943

Figure C-6. The FORTRAN Data Base Creation Program (Sheet 3 of 3)
INDEX

Access control B-1
Actual key file organization 1-3, 3-12, B-1
Advanced Access Methods (AAM) B-1
Alias 2-1, 3-10, B-1
Alternate key
   Listed in sub-schema 2-3, 3-10
   Multiple-index processing 1-3
   Order sequenced 3-14
   READ statement 3-6, 3-12
   START statement 3-6, 3-14
Area (see also Data base file) B-1
ASSIGNID statement 3-18
Automatic recovery 3-18, B-1
Basic Access Methods (BAM) B-1
BEGINTRAN statement 3-16
CDCS
   Definition B-1
   Description 1-2
   Interface
      Establish 3-2
      Terminate 3-4
   Locking mechanisms 4-6
   CDCS Batch Test Facility 1-2, 6-1, B-1
   CDCSBTF control statement 6-1
   Child record occurrence 3-9, 3-13, 3-15
   CLOSE statement 3-4
   COMMITTRAN statement 3-16
   Common item 3-8, 3-10, B-1
   Concurrency 1-4, B-1
Constraint
   Avoiding violations 4-4
   Definition B-1
   Description 1-4, 4-4
Control break
   Definition B-1
   Description 3-13
   Information returned in status block 4-2, 5-12
Control statements
   CDCSBTF 6-1
   DML 5-1
   LDSET 5-1
CYBER Database Control System (see CDCS)
CYBER Record Manager (CRM)
   Definition B-1
   Description 1-3
   Interface 1-3
Data administrator
   Definition B-1
   Description 1-1
   Responsibilities 3-1, 5-1, C-1
Data base B-1
Data base file (see also Realm)
   Accessing 3-1
   Attached by CDCS 3-2
   Creating 3-5, 4-5
   Access Control Methods 3-3
   Error checking 4-4
   File B-2
Lock
   Deadlock situation 4-6
   LOCK statement 3-1
   Use 3-7, 3-8
Manipulating 3-5
Organization 1-3
Position 3-6, 4-3
Privacy 1-4
Processing considerations
   Constraint 4-4
   Deadlock 4-6
   Relation 3-10
Processing function 3-1, 4-3
Status checking 4-4
Data base procedures 1-3, B-1
Data base status block
   Content 4-2
   Definition B-1
   Establishing in FORTRAN program 4-2
   Test for constraint violation 4-4
   Test for control break 4-3
   Test for deadlock 4-6
   Test for null occurrence 4-3
Data base transaction
   Begin transaction 3-16
   Commit transaction 3-16
   Definition B-1
   Drop transaction 3-17
   Example 5-23
   Processing considerations 3-17
   Processing operations 3-15
Data base version 1-4, B-1
Data Description Language (see DDL)
Data item B-2
Data Manipulation Language (see DML)
DML 1-1, B-2
   DEADLE statement 3-8
   Deadlock 4-4, B-2
   DELETE statement 3-9
   Dependent record occurrence 4-4, B-2
   Direct access file organization 1-3, 3-12, B-2
DML statements
   ASSIGNID statement 3-18
   BEGINTRAN statement 3-16
   CLOSE statement 3-4
   COMMITTRAN statement 3-10
   COMMITTRAN statement 3-16
   CLOSE statement 3-10
   COMMITTRAN statement 3-16
   DELETE statement 3-8
   DROPTRAN statement 3-17
   FINDTRAN statement 3-18
   INVOKE statement 3-2
LOCK statement 3-4
OPEN statement
Realm 3-3
Relation 3-10
PRIVACY statement 3-3, 3-10
READ statement
Realm 3-6
Relation 3-12
REWRITE statement 3-7
START statement
Realm 3-6
Relation 3-14
SUBSCHEMA statement 3-1
TERMINATE statement 3-4
UNLOCK statement 3-4
WRITE statement 3-5
DMLDBST routine 4-3
DMS-170
Description 1-1
Feature summary 1-5
Dominant record occurrence 4-4, B-2
DROPTRAN statement 3-17

End option 4-1
End-of-file 3-12, 4-2
EOF (see End-of-file)
ERR option 4-1
Error processing 4-1
Examples
Data Base C-1
FORTRAN application programs 5-3
Sub-schemas 2-1, 3-2, 3-11, C-1
Exclusive locking 3-4, B-2

File (see Data base file)
FINDTRAN statement 3-18
FORTRAN DML (see DML)
FORTRAN source program
Compiling and Executing 5-1
Developing 2-1, 3-1, 5-1
Sample programs 5-3

Hierarchical tree structure 3-9, B-2
Home block B-2

I (mode) 3-3, 3-10
Immediate return 1-4
Indexed sequential file organization 1-3, 3-12, B-2
INVOKE statement 3-2
IO (mode) 3-3, 3-10
Item ordinal 2-2, 4-3

KEY option
READ statement 3-6, 3-12
START statement 3-6, 3-14
Keyword B-2

LDSET control statement 5-1
Listing control 5-3
LOCK statement 3-4
Locking 3-4
Log files
Definition B-2
Used with CDCS 1-4
Used with CDCS Batch Test Facility 6-1
Mapping 2-2, B-2
Master directory
Definition B-2
Sample C-1
Used with CDCS 1-2
Used with CDCS Batch Test Facility 6-1
MODE option
Creating a file 3-5
OPEN statement 3-3, 3-10
PRIVACY statement 3-3
Relation processing 3-10
Multiple index
Processing 1-3
Processor B-2

Null record occurrence
Definition B-2
Description 3-13
Information returned in status block 4-3
Null values 3-5

O (mode) 3-5
OPEN statement
Realm 3-3
Relation 3-10
Operation B-2

Parent record occurrence 3-9, 3-13, 3-15
Permanent file B-2
Primary key
DELETE statement 3-8
Listed in sub-schema 2-2, 3-10
Order sequenced 3-12
READ statement 3-6, 3-13
REWRITE statement 3-7
START statement 3-6, 3-14
Privacy key
Definition B-2, 2-1
Description 1-4
Specification requirements 3-2
PRIVACY statement 3-2, 3-10
Protected locking 3-4, B-2

Rank
Control break status 3-13
Definition B-2
Description 3-9
Null record status 3-13
Returned in status block 4-3
READ
Random 3-6, 3-13
READ statement
Realm 3-6
Relation 3-12
Sequential 3-6, 3-12
Realm (see also Data base file)
Definition B-2
Listed in sub-schema 2-2, 3-1, 3-10
Name returned in status block 4-3
Realm Lock
LOCK statement 3-4
Use when updating 3-7, 3-8
Realm ordinal B-3
Record
Definition B-3
Order stored 1-3
Record Lock 3-7, 3-8
Record occurrence 3-9, 3-12, B-3
Record type B-3
Recovery 1-4, B-3
Relation
   Accessing 3-8
   Definition B-3
   Description 1-4
   Listed in sub-schema 3-10
   Processing considerations 3-10, 3-15
   Processing function 3-9
   Structure 3-9
Relation occurrence 3-8, 3-12, B-3
Restart
   Assigning 3-18
   Identifier 3-18, B-3
   Identifier file 3-18, B-3
   Operation 3-18
   Processing operations 3-18
Restriction
   Definition B-3
   Description 1-4
   Listed in sub-schema 2-3, 3-10
REWRITE statement 3-7
Root realm 3-9, B-3

Schema 1-1, B-3, C-1
Severity error code 4-2

START statement
   Realm 3-6
   Relation 3-14
Status block (see Data base status block)
SUBSCHEMA statement 3-1
Sub-schema
   Definition B-3
   Description 1-1
   Item ordinal 2-2, 4-3
   Samples 2-1
   Used in application programs 3-1, 3-10, C-1
Sub-schema item ordinal 2-2, B-3
Sub-schema library 2-1, B-3
Subroutine 5-9
Subscripting 5-15

TERMINATE statement 3-4
Transaction (See Data base transaction)
Transaction identifier 3-18, B-3

UNLOCK statement 3-4
Updating consideration 3-15

WRITE statement 3-5
This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

___ Please reply      ___ No reply necessary