1

WRDC-TR-90-8007
Volume V
Part 47

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 47 - Embedded SQL User's Manual

K. Stephey, J. Slaton

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 1990

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

[illegible]
DAVID L. JUDSØN, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:

[illegible]
BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

Unclassified

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UM 620341440	5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. V, Part 47
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services 6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI
6c. ADDRESS (City,State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209	7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF 8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533	10. SOURCE OF FUNDING NOS.
11. TITLE (I See block 19	

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
78011F	595600	F95600	20950607

12. PERSONAL AUTHOR(S)
Control Data Corporation: Stephey, K., Slaton, J.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr.,Mo.,Day)	15. PAGE COUNT
Final Report	4/1/87-12/31/90	1990 September 30	80

16. SUPPLEME ARY NOT

WRDC/MTI Project Priority 6203

17. COSATI CODES

FIELD	GROUP	SUB GR.
1308	0905	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify block no.)*

19. ABSTRACT *(Continue on reverse if necessary and identify block number)*

This document explains how to use embedded SQL within applications to access data in an integrated environment.

BLOCK 11:

INTEGRATED INFORMATION SUPPORT SYSTEM
Vol V - Common Data Model Subsystem

Part 47 - Embedded SQL User's Manual

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NO. (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC MTI

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

UM 620341440
30 September 1990

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

SUBCONTRACTOR	ROLE
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface,and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION 1

SCOPE

1.1 Identification

This document contains syntax and instructions for the use of the embedded SQL language used with the Common Data Model precompiler.

1.2 Purpose

The Embedded SQL Precompiler User's Manual is designed to help the application programmer write embedded SQL statements in host languages such as COBOL, C, and FORTRAN, to access the heterogeneous distributed databases supported within the Common Data Model (CDM) operating environment.

1.3 Introduction

The Embedded SQL used by the CDM is adapted from the American National Standard Institute (ANSI) standard, numbered X3H2-89-27, dated November 16, 1988. The Embedded SQL used by the CDM is a subset of the full ANSI SQL Data Manipulation Language specification.

The actual data may be distributed across multiple database management systems (DBMSs) and computers, but the application programmer does not need to be concerned with the actual data locations and actual data formats. These details are handled by the CDM mappings that exist between the schemas.

Embedded SQL allows the application program to interact with the CDM. Embedded SQL uses the three-schema definitions described to the CDM by the Neutral Data Definition Language (NDDL) to satisfy a data request. SQL statements are embedded in the host language programs of COBOL, C, or FORTRAN.

The CDM Fmbedded SQL provides some important functions that standard SQL does not:

a. Referential Integrity; single database and across multiple databases

b. Domain verifications

c. Outer-join operations

1.4 Audience

This manual is intended for application programmers writing SQL applications. It assumes a basic familiarity with SQL and application programming. This manual does not attempt to teach the user to write SQL statements or program using structured programming techniques. The Embedded SQL Reference Manual provides a detailed explanation of the syntax and semantics guiding the development of SQL applications and should be used as a reference.

1.5 How this Manual is Organized

This manual contains four sections and seven appendices as described below:

SECTION 1, SCOPE

This section describes the scope of this manual.

SECTION 2, REFERENCES

This section lists other reference materials and defines commonly used terms, abbreviations, and acronyms.

SECTION 3, INTRODUCTION TO SQL APPLICATION PROGRAMMING

This section includes the options available to the application programmers, design and coding guidelines and the steps necessary to precompile and execute an application.

SECTION 4, FILE INPUT/OUTPUT PRIMITIVES(FIOPs)

This section includes functions available to the application programmer for accessing sequential disk files with fixed length records. These functions provide language and operating system independence.

APPENDIX A, EXTERNAL SCHEMA DATA TYPES

This appendix contains a comprehensive list of External Schema data types for COBOL, FORTRAN, and C programming languages.

APPENDIX B, USER VIEW REPORT

This appendix contains a User View Report.

APPENDIX C, CONCEPTUAL SCHEMA MODEL

This appendix contains a Conceptual Schema IDEF1X model used as an example throughout this manual.

APPEND'X D, SAMPLE COBOL PROGRAMS

This appendix consists of COBOL programs with embedded SQL.

APPENDIX E, SAMPLE C PROGRAMS

This appendix consists of C programs with embedded SQL.

APPENDIX F, SAMPLE FORTRAM PROGRAMS

This appendix consists of FORTRAN programs with embedded SQL.

APPENDIX G, DATE COMPLEX MAPPING ALGORITHMS

This appendix explains the algorithm functions for complex mapping algorithms that support the user in accessing database columns with date data.

SECTION 2

REFERENCES

2.1 Reference Documents

1. UM 620341001, CDM Administrator's Manual, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

2. TBM620341000, CDM1, A IDEF1 Model of the Common Data Model, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

3. IDS150120000C, ICAM Documentation Standards, 23 July 1984, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

4. UM 620341100, Neutral Data Definition Language User's Guide, 31 May 1988, Source: AFWAL/MLTC, Wright- Patterson AFB, Ohio 45433-6533.

5. DS 620341100, Neutral Data Definition Language Development Specification, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

6. PRM620341200, NDML Programmer's Reference Manual, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

7. DS 620341200, NDML Precompiler Development Specification, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

8. DS 620341310, Distributed Request Supervisor Development Specification, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

9. DS 620341320, Aggregate Development Specification, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

10. UM 620344200, Form Processor User's Manual, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

11. UM 620341420, CDM Impact Analysis User's Manual, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

12. UM 620341403, CDM Reports and Application User's Guide, 31 May 1988, Source: AFWAL/MLTC, Wright- Patterson AFB, Ohio 45433-6533.

13. DS 620341420, CDM Impact Analysis Development Specification, 31 May 1988, Source: AFWAL/MLTC, Wright-Patterson AFB, Ohio 45433-6533.

14. PMP/PMS-CDC-R02, Program Master Plan and Program Master Schedule: AAAC Program, 17 June 1988, Source: Control Data Corporation, 2970 Presidential Drive, Fairborn, Ohio 45324.

15. SQL*Forms Designer's Reference, Version 2.0, 1986, Source: Oracle Corporation, Belmont, California.

16. SQL*Forms Operator's Guide, Version 2.0, 1986, Source: Oracle Corporation, Belmont, California.

17. SQL*Report User's Guide, Version 1.0, 1986, Source: Oracle Corporation, Belmont, California.

18. The ORACLE Database Administrator's Guide, Version 5.1, August 19, 1986, Source: Oracle Corporation, Belmont, California.

19. Pro*C User's Guide, Oracle Corporation, Belmont, California, Part #3504-V1.1.

20. JANUS User Guide, Version 2.0, June 1987, Source: D. Appleton Company, Inc., Manhattan Beach, California.

21. ANSI AX3H2-89-001 ISO-ANSI Working Draft "Database Language SQL2", October 1988.

22. ANSI X3.135.1-1989 "Database Language SQL".

23. ANSI X3.168-198X "Database Language Embedded SQL".

24. DB2 Application Programming Guide for TSO and Batch Users, IBM Corporation (Chapter 9).

25. DRAFT SRD-TE-REPORTS, Common Data Model (CDM) Reports: AAAP Program, 29 July 1988, Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

26. DRAFT SRD-TC-FORMS, AAAP-FORMS: AAAP Program: 11 March 1988, Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

27. DRAFT SRD-TA-JANUS, AAAP-JANUS: AAAP Program: 8 April 1988, Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

28. DRAFT SRD-TB-IMPACT, AAAP-IMPACT: AAAP Program: 11 March 1988 Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

29. DRAFT SDS-TB-FORMS, AAAP-FORMS: AAAP Program: 15 July 1988 Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

30. DRAFT SDS-TB-IMPACT, AAAP-FORMS: AAAP-IMPACT: 15 July 1988 Source: Control Data Corporation, 3131 S. Dixie Dr., Dayton, Ohio 45439-2265.

2.2 Terms and Abbreviations

AAAC - Automated Airframe Assembly Center.

AAAP - Automated Airframe Assembly Program.

Application Process - (AP) A cohesive unit of software that can be initiated as a unit to perform some function or functions.

Attribute Use Class - (AUC) An attribute use class indicates that a particular class is used in a particular entity class. The attribute class is either owned by that entity or inherited from another related entity class.

C - A programming language originally developed at AT&T Bell Labs, designed for efficiency, portability and promoting good software engineering practices.

COBOL - A programming language (Common Business Oriented Language) used for business and early data processing applications.

Common Data - all the data of the enterprise. - (CDM) IISS subsystem that describes common data of an enterprise and includes conceptual, external and internal schemas and schema transformations.

Common Data Model - (CDM) IISS subsystem that describes common data of an enterprise and includes conceptual, external and internal schemas and schema transformations.

Common Data Model Administrator - (CDMA) The person or group of persons responsible for creating and maintaining an enterprise's Common Data Model. The CDMA manages the common data rather than managing applications that access data.

Conceptual Schema - (CS) The standard definition used for all data in the CDM. It is based on IDEF1 information modeling.

CS-IS - Conceptual Schema to Internal Schema Mapping.

Cursor - This programmer's concept is the capability in which a procedural programming language interfaces with the SQL set oriented language. SQL retrieves a set of data and the programmer uses a cursor to track the current processing position within that set much as a CRT cursor indicates the user's current position on a terminal screen.

Data Field - (DF) An element of data in the internal schema. Generally, a DBMS will reference data by this name.

Data Item - (DI) An element of data in the external schema. An NDML programmer references data by this name.

Data Type - A specific computer representation of a domain.

DBA - Data Base Administrator.

Distributed Request Supervisor - (DRS) This IISS CDM Subsystem Configuration Item controls the execution of distributed NDML queries and non-distributed updates.

Domain - A logical definition of legal attribute class values.

ES-CS - External Schema to Conceptual Schema Mapping.

External Schema - (ES) An application's view of the CDM's conceptual schema.

Forms - Structured views which may be imposed on windows or other forms. A form is composed of fields where each field is a form, item, or window.

Forms Processor - (FP) A generic set of callable execution time routines available to an application program for form processing.

FORTRAN - A programming language (Formula Translation) used for mathematical and scientific applications.

IDEF1 - A modeling methodology developed under the ICAM program. IDEF1 is one of the three ICAM definition techniques to characterize the manufacturing business environment. IDEF1 is used to produce an "information model" which represents the structure and semantics of information within the environment. IDEF1X is the extended version of IDEF1.

Integrated Information Support System - (IISS) A test computing environment used to investigate, demonstrate and test the concepts of information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems and integration of data resident on heterogeneous databases supported by heterogeneous computers interconnected via a Local Area Network.

Interactive Application Designer - (IAD) A program that performs the functions of the screen painter in SQL*Forms.

Interactive Application Processor - (IAP) A computer program that runs a form.

Internal Schema - (IS) The definition of the internal model, the storage structure definition, which specifies how the physical data are stored and how they can be accessed. It is represented in terms of the physical database components, including record types and inter-record relationships.

JANUS/LEVERAGE - A support tool developed by D. Appleton Company Inc. for data analysis and data modeling.

LISP - List Programming language. This language is used primarily for artificial intelligence applications. It treats program as data, with easy to use list structures.

LOG_UNIT_WORK - A logical unit of work is a recoverable unit. A logical unit of work is a transaction. A recoverable unit has the following properties: (1) it obeys the rules of consistency; (2) it either happens in its entirety or not at all; (3) once it is committed, it cannot be undone.

Mapping - The correspondence of independent objects in two schemas: ES to CS or CS to IS.

Neutral Data Definition Language - (NDDL) A language used to manipulate or populate information in the Common Data Model database.

Neutral Data Manipulation Language - (NDML) A language developed by the IISS project to provide uniform access to common data, regardless of database manager or distribution criteria. It provides distributed retrieved and single node update.

Object - Named Common Data Model item; for example, entity class, data item.

ORACLE - Relational DBMS based on the SQL (Structured Query Language, a product of Oracle Corporation). The CDM is implemented as an ORACLE database.

PL/I - Programming Language One. A procedural programming language originally developed for IBM computers.

PMP&S - Program Master Plan and Schedule.

POSIX - An agreed upon portable operating system interface standard, standardized the interface to many operating system capabilities. Originally developed by IEEE.

Pro*C - An extension of the programming language C that lets you develop user exits and other programs that access the ORACLE database. A Pre-Compiler converts Pro*C code into pure C code.

SQL - Structured Query Language. An agreed upon standard language for relational data base access.

SQL2 - A standards committee successor to the SQL Standards (X3.135-1986 and X3.135-18-1986).

SQLDA - SQL Description Area. An area of program storage used which contains meta-data concerning an SQL statement.

Structured Query Language - (SQL) The primary computer language for manipulating data in the ORACLE database.

Trigger - A sequence of SQL commands and/or SQL*Forms commands that are executed when a certain event occurs.

User exit - A function written in a procedural programming language such as C, which may be invoked by a trigger.

VDT - Video Display Terminal.

SECTION 3

SQL APPLICATION PROGRAMMING

This section introduces you to SQL Application Programming and explains:

- the options available to the application programmer
- how to get started
- design and coding guidelines
- how to precompile, compile, link and execute the application
- how to receive precompilation error/informational messages

Application programs can be developed, written and precompiled on the host machines VAX/VMS™, HP/UX™ and PYRAMID/OSx™. The applications with embedded SQL statements may be written in COBOL, C, or FORTRAN.

The CDM SQL precompiler parses the application program source code and identifies the SQL statements. These statements are transformed from External Schema format to Conceptual Schema to Internal Schema, thereby decomposing the SQL statements to single database requests. These single database requests are transformed to generic DML commands to access the specific databases, either ORACLE® or DB2™ or VAX-11™, to satisfy the request. These programs are known as Request Processors or RPs or RP-Subs.

The SQL commands in the application source program are replaced by host language code which, when executed, activates the run time processes associated with this particular SQL command to evaluate the requests. These generated programs are known as modified USER-APS.

The precompiler also generates Conceptual Schema/External Schema (CS/ES) Transformation programs. These programs transform the final results of a request from Conceptual Schema (CS) format to External Schema (ES) format. The final results are the format in which the application programmer requires data values. The CS/ES program also performs any statistical functions that you request, such as MIN, MAX, SUM, etc., and any presentation request such as ORDER BY or DISTINCT. CS/ES programs are generated only for SQL queries.

The precompiler stores and maintains a cross-reference of all application programs to the External Schema View and data items.

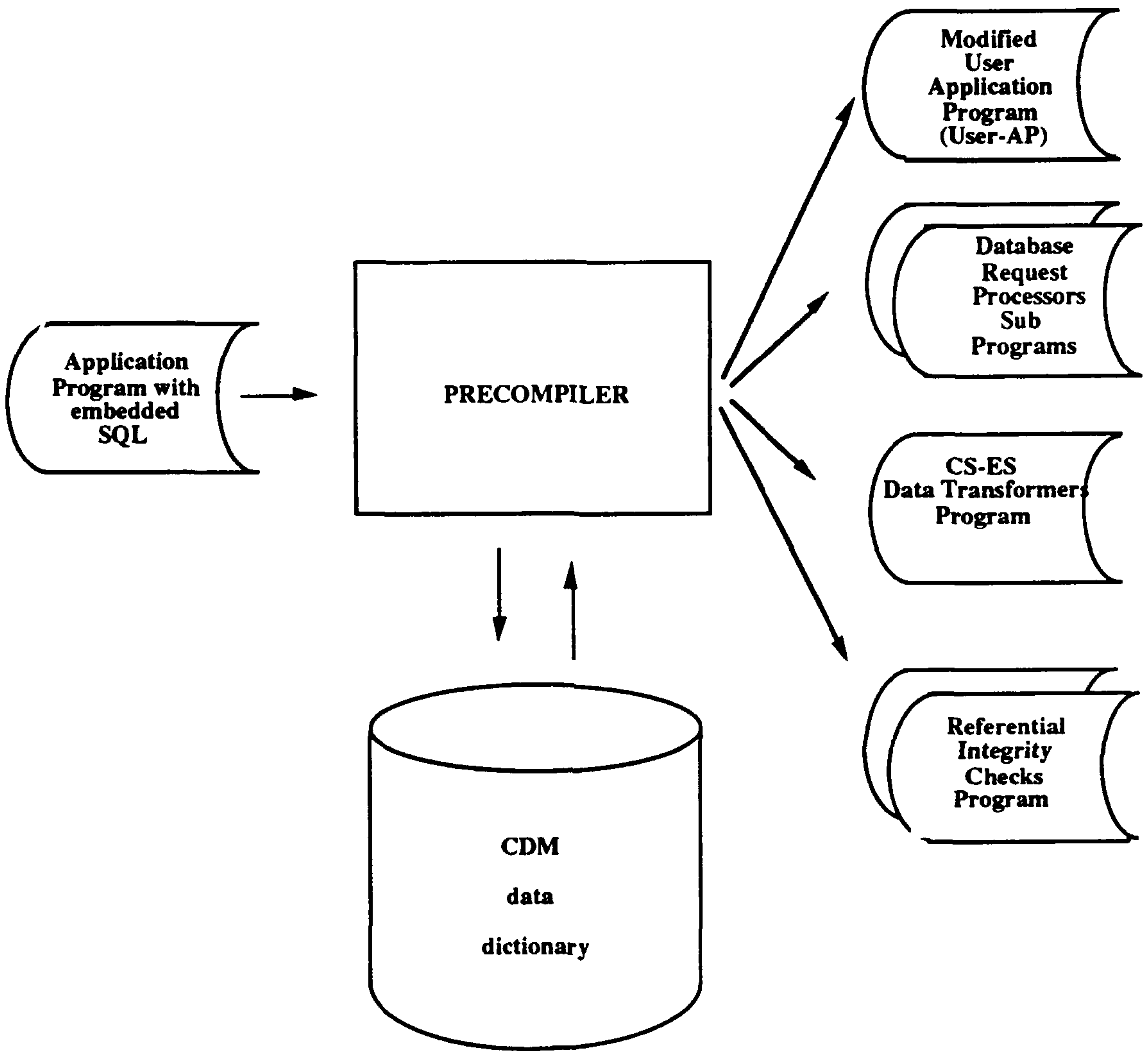
Application Program with embedded SQL
PRECOMPILER
Modified User Application Program (User-AP)
Database Request Processors Sub Programs
CS-ES Data Transformers Program
Referential Integrity Checks Program
CDM data dictionary

Figure 3-1. SQL Precompiler

In addition to heterogeneous database access, the most significant feature of the precompiler is that it generates Referential Integrity Test (RIT) programs for SQL Update (Insert, Update, and Delete) requests. To facilitate your understanding of Referential Integrity, refer to the IDEF1X model presented in Appendix C. The entity and domain RIT programs validate the following activity at execution time:

a. An Insert into a Conceptual Schema entity is performed if its independent entity occurrence exists. For example, a CO_LINE_ITEM instance is represented by a particular CUSTOMER_NO, CUSTOMER_ORDER_NO, and PART_NO among other attributes. A CO_LINE_ITEM instance will be inserted only after validating the existence of this CUSTOMER_ NO, CUSTOMER_ORDER_NO and PART_NO instance. This is a Type-1 Referential Integrity Test.

b. An Insert of a duplicate key value is not performed. For example, the entity PART is uniquely identified by a PART_NO. A part with an identical PART_NO will not be inserted. This is a Key Uniqueness Referential Integrity Test.

c. A Delete of a Conceptual Schema entity is not performed if dependent entity occurrences exist. For example, a CUSTOMER_ ORDER instance will not be deleted, if there are any CO_LINE_ITEM instances associated with this CUSTOMER_ORDER. This is a Type-2 Referential Integrity Test.

d. An Update only of a Conceptual Schema dependent entity key value will be performed only if its independent entity key instance exists. For example, updating PART_NO in the CO_LINE_ITEM instance will be performed only after validating the existence of this PART_NO instance in the PART entity. This is also a Type-1 Referential Integrity Test. Note: update of key attributes is not permitted.

e. An Insert/Update of a data value of an attribute must not conflict with the domain values and ranges specified for that attribute. For example, if the valid domain values for attribute CO_STATUS_CODE were defined to the CDM as being either "O" or "C" or "P", an attempt to insert a data value of "Z" would not be permitted. This is a Domain Verification Test.

Since RITs also access databases, the code generated to test the data is known also as RPs.

Upon successful precompilation, the generated Request Processor programs are transferred to the appropriate machine where the databases are to be accessed. Procedures are initiated to host language precompile and compile the RPs and to compile the modified USER-AP and CS/ES programs.

Prior to linking the user application, another type of RP is generated. This is the RP-Main. The main program performs the functions of logging onto the database, logging off, committing and rolling back operations, and submitting calls to the appropriate RPs to satisfy a request. One RP-Main will be generated for each database to be accessed by an application.

Once the application is linked, it is ready for execution. All applications that access multiple databases or require access to a database that resides on a host machine different than the machine on which it was precompiled will require the services of a network transaction manager and some communications software.

3.1 Options Available to the Application Programmer

The programmer can precompile in these environments:

VMS/VAX 780™
VMS/VAX 8600™
HP-UX/HP-835™
OSx/PYRAMID 98x™

The programmer can embed SQL statements in these host languages:

COBOL
FORTRAN
C

The SQL request can access these databases:

ORACLE
DB2
VAX-11

The databases can reside in these environments:

VMS/VAX
HP-UX/HP-835
MVS/IBM™

Figures 3-2 provide examples of basic SQL verbs.

DATA RETRIEVAL VERBS	**DATA UPDATE VERBS**	**CURSOR COMMANDS**	**TRANSACTION VERBS**
Select	Insert Delete Update	Declare Cursor Open Cursor Fetch Close Cursor	Commit Rollback

Figure 3-2. NDML Verbs

Figure 3-3 provides examples of basic SQL Constructs.

SELECT

```
Select
from
where
 ;
```

INSERT

```
Insert into
(           )
values
(           )
;
```

DELETE

```
Delete

where

;
```

UPDATE

```
Update
set
where

;
```

Figure 3-3. SQL Data Retrieval and Update Constructs

3.2 Getting Started

To create COBOL, FORTRAN, or C programs with Embedded SQL statements, follow the steps provided below.

1. EXTERNAL SCHEMA REPORT

The first thing required is an EXTERNAL SCHEMA or USER VIEW report. This report must contain, at a minimum, a list of views you may write applications against. The report must list all the views and its data items as well as data type of each data item. Refer to Appendix B for an example of a USER VIEW Report. Refer to the CDM Reports User's Manual for all available reports.

2. INTENT OF APPLICATION

You must have a thorough understanding of the scope of your application. You must know:

- What purpose the application serves.
- What data is requested.
- If any specific ordering exists.
- If data is being input. If so, does this view meet all the criteria of an updatable view?

Refer to the Embedded SQL Reference Manual to determine the criteria for an updatable view.

3. LOGISTICS OF APPLICATION

You must know or have answers to these questions:

- What language is the application being coded in?
- What host machine will the application run on?
- What is the logical unit of work name?

4. VIEWS/HOST LANGUAGE DATA TYPE COMPATIBILITY and the CDMA

It is the responsibility of you and the CDMA to ensure that the data type of the data items of your view are compatible with the data types supported by the programming language. Refer to Appendix A for a list of valid data types for each programming language supported by the precompiler.

5. WRITE THE APPLICATION

Use the coding guidelines detailed in the next section while embedding SQL in a host language of COBOL, FORTRAN, or C.

Note: Do not code for Referential Integrity Tests. These tests are generated by the precompiler and validated at execution.

6. LANGUAGE MATRIX

The language of the modified USER-APs and the CS/ES Transform programs always will match the host language of the application program. For example, a COBOL application program generates COBOL modified USER-APs and CS/ES Transformers. You may choose the language you want the RP-Sub Programs to be generated in. Refer to the Request Processor Language/DBMS Matrix in Figure 3-4 for the combinations of language of the RP and DBMSs currently supported by the precompiler. If you do not specify a language, the precompiler selects the same language as that of any RP-Sub precompiled in this logical unit of work or it will default to the language of the application program, provided that language is supported. The precompiler overrides the selection. The RP-Main program will be generated in the same language as all of its RP-Sub programs.

REQUEST PROCESSOR LANGUAGE	DBMS		
	ORACLE INGRES5 INGRES6	VAX-11 IDMS	DB 2
COBOL	YES	YES	YES
FORTRAN	YES	NO	NO
C	YES	NO	NO

Figure 3-4. Request Processor Language/DBMS Matrix

7. COMPLEX MAPPING ALGORITHM DATA TYPES/HOST LANGUAGE COMPATIBILITY

Complex Mapping Algorithms are software modules that you or the CDMA write to perform a complex mapping or transformation. Complex Mapping Algorithms (CMAs) may be written in any programming language. But, when invoked, the data types between the calling program and the CMA must be compatible. The CDMA must ensure that CMAs, if they are to be shared by application programs written in different languages, perform the data type conversion themselves. If the algorithm transforms between an entire record and one or more attributes, the data type of the record structure must be a character buffer. It is highly recommended, that if the CMA is generic (i.e., to be invoked by applications in different languages) then, the data types should also be generic (i.e., alphanumeric or character). In addition, the values must be passed by address, not by value.

8. DATE COMPLEX MAPPING ALGORITHMS

Complex mapping algorithms help you access database columns with date data. See Appendix E for information on using these algorithms.

3.3 Design Guidelines

1. Simple Interfaces

Adhere tc the principle of modular programming: create one module to perform one function. Each routine should consist of one SQL statement. Choose from among Select, Insert, Update, or Delete statements. The routines should have simple interfaces. Call sub-routines to perform complex processing, rather than placing the processing in the SQL program itself. The following input and output parameters shown in Figure 3-5 for each SQL routine are recommended.

SQL STATEMENT	INPUT	OUTPUT
Select	Each search parameter	Each retrieved column Status
Insert	Each column to insert	Status
Update	Each column to modify Each search parameter	Status
Delete	Each search parameter	Status

Figure 3-5. Input and Output Parameters for NDML Programs

2. Logical Unit of Work

Group applications so that all the work performed by the application can be committed together or rolled-back together to maintain database consistency. Group all routines affecting the same tables into one logical unit of work.

Note: A SQL routine can only participate in one logical unit of work.

3. Referential Integrity

Do not code for Referential Integrity and Domain verifications.

4. Distributed Data

Do not design or code the application as if the data were distributed. The application programmer codes against a neutral view of data. The precompiler will take care of transforming the request to the required databases and machines.

5. One-Row versus Multi-Row Retrieval

Distinguish if the query will return one row or many rows. Examples of single row retrievals are statistical functions or retrieval of an employee name given the employee number. An example of a many row retrieval is the selection of all line items in an order. SQL provides syntax to allow for both types of retrieval.

6. Insert, Update, and Delete

SQL Update routines should only perform the SQL action. It is the calling routine's responsibility to determine if the program has an error when a referential integrity test fails. This would provide for shareable update routines.

7. Nested Queries versus Joins

For performance reasons, use joins and outer joins to join across views rather than using nested queries.

8. Program Life Cycle

Design programs for maintainability rather than performance to provide for a longer life cycle.

3.4 Coding Guidelines

1. Templates

Use coding templates to insure consistency and compliance with coding standards.

2. User View Report

An EXTERNAL SCHEMA Report or a USER VIEW Report is necessary when writing a SQL application to allow you to accurately specify the data items and data types.

3. Data Type Compatibility

The data type of each data item in the USER VIEW Report must correspond to the variable definition in the SQL application program.

The following is a list of the valid, generic data types for a statistical query. Refer to Appendix A for the specific data type supported by the programming language.

STATISTICAL FUNCTION	GENERIC DATA TYPE
AVG	numeric
COUNT	numeric and character
SUM	numeric
MIN	numeric and character
MAX	numeric and character

Figure 3-6. Data Types for Statistical Functions

4. Copy and Include Facilities

Use Copy and Include facilities for commonly used table and data structure definitions.

5. SQL Status Checking

The precompiler always defines and updates a program variable to indicate the execution status. Check this status after each SQL statement.

Figure 3-7 shows error code values:

ERROR CODE	EXPLANATION
100	no data
0	no error
<0	error
-49901	failure of a type 1 referential integrity test on an Insert or Modify
-49902	failure of a type 2 referential integrity test on a Delete
-49903	failure of a key uniqueness test on an Insert
-43306	failure of a domain verification module

Figure 3-7. SQL Error Codes

The error codes will be found in SQLCODE (or SQLCOD for FORTRAN and sqlca.sqlcode for C) after every SQL statement. This variable is generated into the user program.

6. Commit and Rollback

It is your responsibility to commit or rollback the effects of the SQL update. If a commit is not executed, the changes will not be committed to the database. You may use the SQLCODE variable to determine whether to commit or rollback a transaction.

7. Termination

The driver program of a COBOL, C, or FORTRAN application must insert a call to the routine "TRMDML." This routine logs off from the database.

8. Compile before Precompile

Remove host language compilation errors be.'ore precompilation. The following sections discuss compilation of user-written Embedded SQL application programs.

9. Embedded SQL Reference Manual

All SQL application programmers are advised to consult the Embedded SQL Reference Manual.

3.5 Embedding SQL in COBOL

COBOL compilers do not know the syntax or meaning of the SQL commands. Therefore, a COBOL application program that contains SQL statements must be precompiled and compiled by the CDM precompiler. The precompiler substitutes COBOL code into the application program in place of the SQL statements. The substituted code provides the mechanisms necessary to activate the run time processes to evaluate the request.

To allow the precompiler to distinguish SQL statements from COBOL statements, each SQL statement must begin with EXEC SQL and end with END-EXEC; the rest of the SQL statement must begin in column 8 or greater. After precompilation, all SQL lines will appear as comment lines to the COBOL compiler.

During precompilation, code is generated into the users application program. For a COBOL application, code is generated in the Working-Storage Section and Procedure Division. A list of reserved variable names and label statements for any COBOL application program used in the CDM environment is shown below. These reserve terms are valid COBOL variables and statements, but cannot be used as user-defined variables or labels.

Any variable names beginning with:

APL
CDM
CS
CSQ
ES
ESQ
FCB
JQG
KES
NDML
RES
RFT

If using a cursor, cannot have a variable declared as cursor-name-FLAG.

Any label names such as:

BREAK-NDML-nn
CDM-EXIT-nn
CDM-LOOP-nn
CD-LOOP-nn
CE-LOOP-nn
END-NDML-nn
RESULT-nn

where nn depends on the nesting structure of the embedded SQL query statements and the number of SELECT statements in the application program.

And the following variables also should not appear as user-defined variables or labels:

DISPOSITION	RIT-MESG
FILE-OPEN	SQLCODE
HOLD-STATUS	USER-MOD
NUMBER-OF-RECORDS	

Additionally, COBOL Copy library include members CHKCDM and ERRCDM will be included in the application's Working Storage Section.

Refer to the Embedded SQL Reference Manual for additional information on SQL Retrieval and Update Statements.

3.6 Embedding SQL in FORTRAN

FORTRAN compilers do not know the syntax or meaning of the SQL commands. Therefore, a FORTRAN application program that contains SQL statements must be precompiled by the precompiler. The precompiler substitutes FORTRAN subroutine calls into the application program in place of the SQL statements. These subroutines provide the mechanisms necessary to activate the run time processes to evaluate the request.

To allow the precompiler to distinguish the SQL statements from FORTRAN statements, the first line must start with EXEC SQL in column 7 and any additional lines must have a non-blank and non-zero in column 6 and have the rest of the SQL statement begin in column 7 or greater. Lack of a non-blank and a non-zero in column 6 indicates termination of SQL statement. After precompilation, all SQL lines will appear as comment lines to the FORTRAN compiler.

During precompilation, code is generated into the application program. This generated code consists of variable definition statements and formatted input/output statements. Following is a list of reserved variable names and CONTINUE statement numbers for any FORTRAN application process used in the CDM environment. These are legal FORTRAN variables and statements but must not appear in the original application program as user-defined variables or labels.

Variable Names:

CDMxxx	CEFLAG	CHARCT
CSCNTC	CSCNTI	CSUSED
DECIML	DIGIT	DRSACT
EOFFLA	ENPOSC	ENPOSR
FILEST	FLAGAR	NDMLCT
NDMLST	NOSSUB	NUMREC
PTRCDM	RFTCTC	RFTCTI
RFTCTJ	SIGN	SPOSC
SPOSR		

where xxx is any letter from A to Z.

Any CONTINUE Statements within the range 91000 to 99999 also must not be defined as variables or labels.

User-defined variables should be limited to six characters.

On the HP, if your program calls non-FORTRAN programs, write an $ALIAS statement for each non-FORTRAN program called. For example, $ALIAS MRTN C (%REF())

Refer to the Embedded SQL Reference Manual for additional information on SQL Retrieval and Update statements.

3.7 Embedding SQL in C

C compilers do not know the syntax or meaning of the SQL commands. Therefore, a C application program that contains SQL statements must be precompiled by the precompiler. The precompiler substitutes C subroutine calls into the application program in place of the SQL statements. These subroutines provide the mechanisms necessary to activate the run time processes to evaluate the request.

To allow the precompiler to distinguish the SQL statements from C statements, the SQL command must begin with EXEC SQL and terminate with ";".

During precompilation, code is generated into the user's application program. This generated code consists of variable definition statements and formatted input/output statements. Following is a list of reserved variable names and CONTINUE statement numbers for any C application process used in the CDM environment. These are legal C variables and statements, but must not appear in the original application program as user defined variables or labels.

The following variable names:

cdmxxx	ceflag	charct
cscntc	cscnti	csused
deciml	digit	drsact
eoffla	enposc	enposr
filest	flagar	ndmlct
ndmlst	nossub	numrec
ptrcdm	rftctc	rftcti
rtfctv	sign	sposc

sposr	tmpnum	tmpnumZ

where xxx... is any letter from A to Z

Any label names such as:

BREAK_NDML_nn
CDM_EXIT_nn
CDM_LOOP_nn
CD_LOOP_nn
CE_LOOP_nn
END_NDML_nn
RESULT_nn

where nn depends on the nesting structure of the embedded SQL query statements and the number of SELECT statements in the application program.

The application programmer is advised to define user variables limited to eight characters.

Refer to the Embedded SQL Reference Manual for additional information on SQL Retrieval and Update statements.

3.8 Precompile Compile, Link, and Execute the Application

You can precompile applications in the UNIX operating system environment. The procedure you use, GENAP, is implemented as a Bourne shell script under UNIX.

GENAP offers menu options to precompile, compile, and link applications containing SQL source code statements. It offers an interactive (menu-driven) mode.

The utility "cdmsetup" must be executed before running "genap." This sets up various environmental variables used by GENAP.

GENAP is started by entering "genap" on the command line or selecting GENAP from the CDM TOOLS Main Menu.

```
***APPLICATION GENERATOR MAIN MENU***

        1.    Precompile Only

        2.    Load (Link) Only

        3.    Precompile and Link

        4.    Help - Option Descriptions

        5.    Exit
```

In the following description of these options, the term "PRC file" refers to a COBOL, C, or FORTRAN file with embedded statements. Such a file will have a .PRC or .prc suffix on its name. IISSGLIB is the name (set up as an environmental variable) of the object library where the compiled modules will be archived.

1. Precompile Only

If you choose this option, you will be asked to enter the name of your logical unit of work, the name of the host where your application will run, your CDM username/password, the language of the database requests, language of the PRC, embedded language (enter SQL), and whether or not you want your obsolete generated code deleted. Then you will be asked to name each PRC you want precompiled. Your PRC file(s) will be run through the precompiler. You will be notified if there are any errors in the precompile and the process will stop. If the precompile is successful, the generated code will be compiled and placed in IISSGLIB.

2. Load (Link) Only

If you choose this option, you will be asked to enter the name of your logical unit of work, the name of the driver program, the name of the host where your application will run, language of the PRC, embedded language (enter SQL), and your CDM username/password. An RP Main will be generated, checked for errors, compiled, and placed in IISSGLIB. Pending successful completion of that, your application will be linked. Your driver must have previously been compiled and the object must be in the directory where you are running GENAP from.

3. Precompile, Compile, and Load

If you choose this option, you will be asked to enter the name of your logical unit of work, the name of the driver program, the name of the host where your application will run, your CDM username/password, the language of the database requests, whether or not you want your obsolete code deleted, and the name of each PRC that you want precompiled. The actions carried out by this step are a combination of the actions in steps 1 and 2. A brief summary is: Precompile, compile all generated code and put it in IISSGLIB, generate the RP Main, compile it and put it in IISSGLIB, and link the application.

4. Display Help Screen

5. Exit

3.9 Example of Precompiling in the UNIX Environment

The following is an example of precompiling and linking an Application Program. The application program is part of the logical unit of work DEMO, whose source code can be found in the file CUSTORD.PRC. The program CUSTORD is invoked by the driver program CUSTDR. There must be a driver program to start the program. CUSTDR is a COBOL AP, although similar procedures are used for FORTRAN or C programs. The linked AP is called custdr.exe. Before running GENAP, read the comments at the beginning of the procedure file.

```
$ genap

          ***APPLICATION GENERATOR MAIN MENU***

     1.   Precompile Only
     2.   Load (Link) Only
     3.   Precompile and Link
     4.   Help - Option Descriptions
     5.   Exit

          PLEASE ENTER AN OPTION NUMBER:  3
  Enter Name of Logical Unit of Work:  demo
  Enter Name of Host Where Application Will Run:  hp
  Enter the Language of RP_SUB:  cobol
  Enter language of Source Program(s) (C/FORTRAN/COBOL):  cobol
  Enter the type of Embedded language used (NDML/SQL):  sql
  Enter Your CDM Username/Password:  cdm/cdm
  Enter Name of the Application:  custdr
  Do You Want Obsolete Generated Code Deleted (Y/N):  y
  Enter Name of PRC (C/R To Stop, Leave .PRC Off):  custord
  Enter Name of PRC File (C/R To Stop, Leave .prc Off):  <CR>

Beginning precompile

SQL PRECOMPILE SUCCESSFULLY COMPLETED

ALL GENERATED CODE SUCCESSFULLY COMPILED

Beginning Generation of RP-Main

GENERATION OF REQUEST PROCESSOR MAIN COMPLETE

ORACLE Precompiling the RP-Main
COBOL Compiling the RP-Main

Beginning Link

LINKING COMPLETED
```

3.10 Example of Executing in the UNIX Environment

```
$ custdr.exe
```

3.11 Receiving Precompilation Error/Informational Messages:

After precompilation, you can obtain status information. If the precompilation was successful, the procedure file GENAP continues to compile the generated code resulting from all the successful precompilations. Applications precompiled under a logical unit of work must all be successfully precompiled before an executable can be linked.

You can find errors or warning messages in two files in the UNIX environment:

errlog.dat
luwname.err

where *luwname* is the name of the logical unit of work for the User Application that you specified to GENAP.

Errors encountered and logged in errlog.dat are fatal or system errors. Refer the problem to the system administrator. These errors usually occur when:

The ORACLE instance is inactive
The CDM database has incorrect meta data

Messages in the .err file are either informational, syntax, or semantic errors.

All precompilation syntax and semantic errors must be corrected and the failed application program must be re-submitted for precompilation. Only the programs that did not precompile successfully need to be re-precompiled. The precompiler allows for separate precompilation. Upon re-precompilation, if you replied "Y" to the option "Do you want obsolete Generated Code Deleted", these generated modules will be deleted from the generated object code library IISSGLIB.

SECTION 4

FILE INPUT/OUTPUT PRIMITIVES (FIOPS)

The File Input/Output Primitives (FIOPS) provide a generic transportable interface to access system specific files allowing language and operating system independence for file manipulations.

This section is included for the user manipulating large amounts of data. For that situation, files may be used to insert from and select into. These primitives are recommended for ease of use when reading or writing the files.

The FIOPS are a group of functions to allow programs a generic interface to access system specific files. The functions are:

- a temporary file name
- open file
- read record
- write record
- seek forward
- seek of backword
- close file
- sort/merge

The FIOP functions are limited for use with *sequential disk files with fixed length records*. The functions cannot be used with tape files.

4.1 Parameter Formats

Character strings are arrays of characters with fixed maximum lengths, terminated by a NUL character ('LOW-VALUE' in COBOL, '\0' in C) if less than the maximum length. Address parameters are pointers to variables, and integer parameters are binary signed integers, at least 32 bits wide (PIC S9(9) COMP in COBOL, long int in C).

Note that in the C examples to follow, character arrays are one longer than needed, and a NUL is assigned to the last character for ease of use in C functions. This is not necessary, but many C string functions require a terminating NUL. As a reminder, arrays in C start with element zero (0) and the last subscript is one less the declared size.

4.2 Status Return Codes/Error Handling

Status return codes are defined as strings (arrays) of five characters. A return code of all zeros ("00000") indicates successful completion of the called function, otherwise an error has occurred. The error codes returned are formatted in such a way that one can distinguish the subsystem, function being performed, and determine what kind of error occurred. The FIOPs (like other primitives) will call ERRPRO with an accompanying descriptive message to be put in the ERRLOG file if the error is fatal. If the error code is a warning, ERRPRO will not be called.

The first two digits of the returned error codes for FIOPs are defined to be "11" (one-one). The left "1" indicates "Primitives", and the right "1" shall indicate "File I/O" primitives. The third digit will refer to the FIOP function performed, and the last two digits will refer to function specific errors. Further, the last digit will indicate the severity of the error: zero will be for warnings, and nonzero will be for fatal errors. This will allow the programmer to easily determine a fatal/nonfatal status.

4.3 NAMFIL - Temporary File Name Function

Create and return a name for a file, suitable for use with the Open function. These file names will not match any currently existing on the the computer.
Usage Format:

COBOL Example:

```
01  FILE-NAME          PIC X (80).
:
:
    CALL "NAMFIL" USING FILE-NAME.
```

C Example:

```
char file_name[81];
:
file_name [80] = '\0';
namfil (file_name);
```

Parameter:

File Name: A fully-qualified file name returned in local system format for a temporary file. The receiving variable should be at least 80 bytes long. If the filename is less than 80 characters, it will be followed immediately by a NUL, and the rest of the field will be underlined. If an error occurs, the first position of this field will be returned as NUL.

4.4 OPNFIL - File Open Function

Prepare the file for access, initialize the file control block (FCB), and allocate space for the file, if needed. The file status will be defined as "share" for input mode, and "exclusive" for output or append modes. If a nonexistent file is opened for append access, it will be created. In the unusual case that a file is created for append, then a valid record length must be passed to OPNFIL.

Usage Format:

COBOL Example:

```
01   FILE-NAME               PIC X(80)        VALUE SPACES.
01   FILE-CONTROL-BLOCK1     PIC S9(9) COMP   VALUE ZERO.
01   RET-STATUS              PIC X(5)         VALUE SPACES.
     88 STATUS-OK                             VALUE "00000"
     88 NEW-APPEND-FILE                       VALUE "11130".
01   STATUS-RDF REDEFINES     RET-STATUS.
     05 FILLER                PIC X(4).
     05 SEVERITY              PIC X.
        88 WARNING                            VALUE "O".
```

```
01   ACCESS-MODE              PIC  X            VALUE "W"
01   RECORD-LENGTH            PIC S9(9) COMP    VALUE ZERO.
01   NUMBER-OR-RECORDS        PIC S9(9) COMP    VALUE ZERO.
:
:
     MOVE (expression) TO RECORD-LENGTH.
     MOVE (expression) TO NUMBER-OF-RECORDS.
     CALL  "OPNFIL" USING FILE-CONTROL-BLOCK1, RET-STATUS,
                         FILE-NAME, ACCESS-MODE, RECORD-LENGTH,
                         NUMBER-OF-RECORDS.
     IF NOT WARNING
        PERFORM  ... (fatal status processing).
```

C Example:

```
#define GDSTAT      "00000"
#define WARNING     '0'
#define SEVERITY    status [4]
int  *fcb;
char status [6], file_name [81];
long rec_lth, num_recs;
:
/* Make sure string is NUL terminated */
status [5] = '\0';
:
rec_lth = (expression);
num_recs = (expression);
opnfil(&fcb, status, file_name, "R", &rec_lth, &num_recs);
If (SEVERITY !=WARNING )
{    ...error processing;    }
```

Parameters:

File Control Block (FCB): The address returned to the caller when an FCB is created and initialized by Open. The FCB will have to be passed to the other FIOPs by the caller. The format of the control block is system dependent and the user need not be concerned with its format. A separate FCB is required for each file open at the same time.

Status: A generic code returned from this function which is five characters long. Possible codes are:

"00000" - SUCCESS
"11101" - INVALID PARAMETER
"11102" - FILE OPEN ERROR
"11103" - FILE NOT FOUND
"11104" - FILE NOT CREATED
"11105" - READ ACCESS DENIED
"11106" - WRITE ACCESS DENIED
"11107" - FILE NOT SEQUENTIAL
"11130" - CREATED NEW FILE FOR APPEND

File Name: A fully-qualified file name in local system format. A local file name is an array of characters, and if less than the maximum length is terminated by a space and/or NUL character. This filename may be user specified or one returned from NAMFIL.

Access Mode: Indicates whether the file is opened for reading (input only), writing (output only, all prior data lost), or append (output to the end of the file only, all prior data preserved). Legal values are "R", "W", and "A" for Read, Write, and Append.

Record Length: An integer number indicating the maximum size of the records in the file. The record buffer should be at least this size. This value is user specified for output and append access accuracy (or used as the required length to create a file) in append mode.

Number of Records: An integer that is the estimated total number of records for an output or append file given by the user. This parameter will allow OPNFIL to make sure that enough space exists for the file, and possibly to reserve it for use by this process. This parameter will not be used for Read access. The FIOPs will attempt to give the user more space for a file if required.

4.5 INPFIL - File Read Function

Retrieves data from an open file. Successive calls will read through the file in sequential order. This function will do record input only, so that if the size of the actual record is greater than the supplied buffer length, the record is truncated to this length. This should help prevent other data in the program from being inadvertently overwritten. To move the record pointer to the previous or next record without reading, use the seek function.
Usage Format:

COBOL Example:

```
CALL "INPFIL" USING FCB-X, RET-STATUS, RECORD-BUFFER,
                    BUFFER-LENGTH, RETURN-LENGTH.
```

C Example:

```
recbuf_lth = sizeof(rec_buf);
inpfil(&fcb_x, status, rec_buf, &recbuf_lth, & rtn_lth);
```

Parameters:

File Control Block (FCB): A user supplied address, returned previously by the Open routine.

Status: A generic code as mentioned previously. Possible codes are:

"00000" - SUCCESS
"11201" - INVALID PARAMETER
"11202" - FILE NOT OPENED FOR INPUT
"11203" - READ ERROR
"11210" - END OF FILE
"11240" - RECORD TRUNCATED

Record Buffer: A buffer where the record data will be stored. The record length was previously returned by the Open call. The returned data is not NUL terminated.

Buffer Length: The user specified integer length of the buffer used for the returned record. If the user gives a length smaller than the record length returned by the Open function, the data will be truncated.

Return Length: The actual integer size of the record placed in the buffer by this function. It should always be equal to the record length returned by the Open function for fixed record length files, unless the buffer length is smaller than the record size, in which case the return length will be set to the buffer length. If the buffer length is smaller than the actual record length, the data will be truncated and an appropriate warning status code returned.

4.6 OUTFIL - File Write Function

Add a new record to a file opened for output or append. The record size must be the same as that referenced in the Open function. File overflow will not be allowed, a "file full" error will be returned instead.
Usage Format:

COBOL Example:

```
CALL "OUTFIL" USING FCB-X, RET-STATUS, RECORD-BUFFER, RECORD-
                         LENGTH.
```

C Example:

```
outfil(&fcb_x, status, rec_buf, &rec_lth);
```

Parameters:

File Control Block (FCB): A user supplied address, returned previously by the Open routine.

Status - A generic code as mentioned previously. Possible codes are:

"00000" - SUCCESS
"11301" - INVALID PARAMETER
"11302" - FILE NOT OPENED FOR INPUT OR APPEND
"11303" - FILE FULL
"11304" - WRITE ERROR
"11305" - RECORD TOO LARGE, NOT WRITTEN

Record Buffer: The record will be taken from this buffer and written to the file. The data in this record is not NUL terminated, and is assumed to be the length returned by the previous Open function.

Record Length: The integer length of the record. This parameter will be ignored for the current implementation of fixed length records. It will be the length determined in the Open function. Variable length output may be a future enhancement.

4.7 SEKFIL - File Seek Function

Repositions to a different record in the file relative to the current file position. This will allow very limited non-sequential access. Only two possibilities will be supported, seek to the next record and seek back one to the previous record. It is defined that a read moves the file position pointer to the beginning of the next record so that a subsequent forward seek will skip a record. Seek is only valid for files opened for read.
Usage Format:

COBOL Example:

```
MOVE 1 TO RECORD-COUNT.
CALL "SEKFIL" USING FCB-X, RET-STATUS, RECORD-COUNT.
```

C Example:

```
rec_cnt =1:
sekfil(&fcb_x, status, &rec_cnt);
```

Parameters:

File Control Block (FCB): A user supplied address, returned previously by the Open routine.

Status: A generic code as mentioned previously. Possible codes are:

"00000" - SUCCESS
"11401" - INVALID PARAMETER
"11402" - FILE NOT OPENED FOR INPUT
"11410" - END OF FILE
"11420" - BEGINNING OF FILE

Record Count: This is a signed integer number of records to reposition in the file. Only two values will be accepted. These values are +1 for seeking forward one record and -1 to go back one record.

4.8 CLSFIL - File Close Function

Terminate file access, cleanup (release resources, etc.), keep or delete the file as user specified.

Usage Format:

COBOL Example:

```
CALL "CLSFIL" USING FCB, RET-STATUS, DISPOSITION.
```

C Example:

```
clsfil(&fcb_1, status, "K");
```

Parameters:

File Control Block (FCB): A user supplied address, returned previously by the Open routine.

Status: A generic code as mentioned previously. Possible codes are:

"00000" - SUCCESS
"11701" - INVALID PARAMETER
"11702" - FILE NOT DELETED
"11703" - FILE NOT CLOSED

Disposition: A one character string which indicates file disposition. There are three options: "K", "P", and "D" for Keep, Pass, and Delete. A file closed with keep disposition will be saved to disk. A file closed with Pass will be helpful (possibly in memory) for future access until another process deletes it or makes it permanent. Delete causes the file to be deleted. If this parameter is not specified (left blank or NUL) then Pass will be assumed.

4.9 SRTFIL - Sort/Merge Function

Take one or more files, merging if more than one, and order the records according to the users' directions. The result will be put in an output file (name supplied by the user) different from any of the input file(s). The user is responsible for making the fields in the input files compatible. The sort routine may pad records so that all output records are the same length.
Usage format:

COBOL Example:

```
CALL "SRTFIL" USING IN-FILE-NAMES, OUT-FILE-NAME, SORT-KEY,
               OPTIONS, OUT-RECORD-COUNT, RET-STATUS.
```

C Example:

```
srtfil(in_fnames, out_fname, sort_key, options, &out_cnt, status);
```

Parameters:

Input filenames: A string containing one to four filenames (multiple filenames are separated by a plus sign) indicating the file(s) to be sorted. The total length of this field will be a maximum of 324 characters. There must be a terminating NUL character after the last input filename given. Extra spaces around separate filenames will be ignored.

Output filename: A string containing one filename in the same format shown for the Temporary Filename and Open functions.

Sort key description: A string describing the sort key. For each field chosen to sort upon, there will be four subparameters in the sort key separated by commas. The sort control groups (sets of four subparameters), if more than one, must be specified from highest priority to lowest and are also separated by commas. There shall be no limit on the number of sort control groups, except that the maximum length of the sort key description string will be 2000 characters. The end of the specifications will b[illegible]licated by either a zero in subparameter 1, a blank, or a NUL character. The subparameters are: (1) starting position of the field, (2) length of the field in bytes, (3) type of field, and (4) the sort order. The starting position and field length are positive integer numbers. The field type is a two-character field and has valid values of:

CH - character, unsigned
BI - binary, unsigned
PD - packed decimal, signed
DT - signed decimal, trailing overpunch

The sort order is either A for Ascending, or D for Descending. An example of a sort key description might be:

"1,8,CH,A,9,4,PD,D"

which indicates that the first sort field is composed of the first eight characters of the record in ascending order and then a four byte packed decimal field starting at the ninth character in descending order.

Note: If the first character of the sort key descripton is NUL, blank, or zero, a file copy (concatenate if more than one input file) will be assumed.

Sort options: The sort options string will be reserved for future use if required. Specific defaults will be assumed in this implementation. They are: (1) the character collating sequence will be the computer's native code, ASCII on the VAX, and EBDCIC on the IBM, (2) records with duplicate keys will be kept in the resultant file, (3) the order of equally collating records is not guaranteed to be preserved from input to output.

Output record count: This is the integer number of records that the output file contains upon successful sort completion.

Status: A generic code as mentioned previously. Possible codes are:

"00000" - SUCCESS
"11901" - INVALID PARAMETER
"11902" - SORT PACKAGE ERROR
"11903" - ERROR CLOSING INPUT FILE(S)
"11904" - ERROR CLOSING OUTPUT FILE
"11905" - ERROR OPENING INPUT FILE(S)
"11906" - ERROR OPENING OUTPUT FILE
"11907" - ERROR READING FILE(S)
"11908" - INSUFFICIENT WORK SPACE
"11909" - ERROR WRITING FILE

4.10 Data Organization

The following table is a summary description of the parameters used by the FIOP functions. Other data elements within each of the VAX and IBM specific routines are not included here.

PARAMETER	TYPE	LENGTH/RANGE	USED BY
ACCESS-NODE	string	1 character	OPNFIL
BUFFER_LENGTH	integer	1 to rec.lth.	INPFIL
DISPOSITION	string	1 character	CLSFIL
FCB	pointer		OPNFIL, INPFIL, OUTFIL, SEKFIL, CLSFIL
FILENAME	string	80 characters	NAMFIL, OPNFIL
IN-FILE-NAMES	string	324 characters	SRTFIL
NUMBER-OF-RECORDS	integer	>0	OPNFIL
OPTIONS	string	not used	SRTFIL
OUT-FILE-NAME	string	80 characters	SRTFIL
OUT-RECORD-COUNT	integer	>=0	SRTFIL
RECORD-BUFFER	string	user-specified	INPFIL, OUTFIL
RECORD-COUNT	integer	+1, -1	SEKFIL
RECORD-LENGTH	integer	>0	INFIL, OUTFIL
RETURN-LENGTH	integer	>0	INPFIL
SORT-KEY	string	2000 characters	SRTFIL
STATUS	string	5 characters	OPNFIL, INPFIL, OUTFIL, SEKFIL, CLSFIL, SRTFIL

APPENDIX A

EXTERNAL SCHEMA DATA TYPES

This appendix consists of the External Schema data types available for COBOL, C, and FORTRAN programmers.

An application programmer writing a COBOL program may define:

a Character string	(C)	as	PIC X(n)
a Variable character	(V)	as	PIC X(n)
a Signed number	(S)	as	PIC S9(n) v9(n) SIGN LEADING SEPARATE
a Decimal number	(D)	as	PIC S9(n) v9(n) SIGN LEADING SEPARATE
a Numeric number	(M)	as	PIC S9(n) v9(n) SIGN LEADING SEPARATE
a Packed value	(P)	as	PIC S9(n) V9(n) COMP-3
an Unsigned number	(N)	as	PIC 9(n) V9(n)

An application programmer writing a FORTRAN program may define:

a Character string	(C)	as	CHARACTER*N
a Variable character string	(V)	as	CHARACTER*N
an Integer number	(I)	as	INTEGER
a Small Integer number	(A)	as	INTEGER
a Floating point number	(F)	as	DOUBLE PRECISION
a Real number	(R)	as	REAL *4

An application programmer writing a C program may define:

a Character string	(C)	as	char (n)
a Variable character string	(V)	as	char (n)
a Small Integer number	(A)	as	long
an Integer number	(I)	as	long
a Double Precision number	(O)	as	double
a Floating Point number	(F)	as	double

APPENDIX B

USER VIEW REPORT

Tue Nov 8 page 1

External Schema Report

View Name	Data Item Name	Data Type Name	Data Type	Size	#Dec
PART	PART_NO	NUMBER	C	15	0
	PART_DESC	DESC	C	30	0
	PART_UNIT_OF_MEASURE_CD	CODE	C	2	0
	PART_NORMAL_LEAD_TIME	QUANTITY	N	8	2
	PART_UNIT_COST	DOLLARS	N	8	2
CUSTOMER_ORDER	CUSTOMER_NO	ORDER_NO	C	10	0
	CUSTOMER_ORDER_NO	ORDER_NO	C	10	0
	CO_PURCHASE_ORDER_DATE	DATE1	C	6	0
	CO_STATUS_CODE	CODE	C	2	0
CO_LINE_ITEM	CO_LINE_NO	NUMBER	C	15	0
	COLI_STATUS_CODE	CODE	C	2	0
	CO_DUE_DATE	CODE	C	6	0
	CO_QUANTITY_REQUIRED	DATE1	N	8	0
	CUSTOMER_NO	QUANTITY	C	10	2
	CUSTOMER_ORDER_NO	ORDER_NO	C	10	0
	PART_NO	NUMBER	C	15	0
PART_F	PART_NO	NUMBER	C	15	0
	PART_DESC	DESC	C	30	0
	PART_UNIT_OF_MEASURE_CD	CODE	C	2	0
	PART_NORMAL_LEAD_TIME	QUANTITY	N	8	2
	PART_UNIT_COST	DOLLARS_F	F	8	2
CO_LINE_ITEM_F	CO_LINE_NO	NUMBER	C	15	0
	COLI_STATUS_CODE	CODE	C	2	0
	CO_DUE_DATE	DATE1	C	6	0
	CO_QUANTITY_REQUIRED	QUANTITY_F	F	8	2
	CUSTOMER_NO	ORDER_NO	C	10	0
	CUSTOMER_ORDER_NO	ORDER_NO	C	10	0
	PART_NO	NUMBER	C	15	0

APPENDIX C

CONCEPTUAL SCHEMA MODEL

This appendix consists of a portion of a Conceptual Schema IDEF1 Model against which NDDL user views have been developed. The appendix also consists of a USER VIEW REPORT against which all SQL applications in this manual have been written.

Conceptual Schema IDEF1 Model

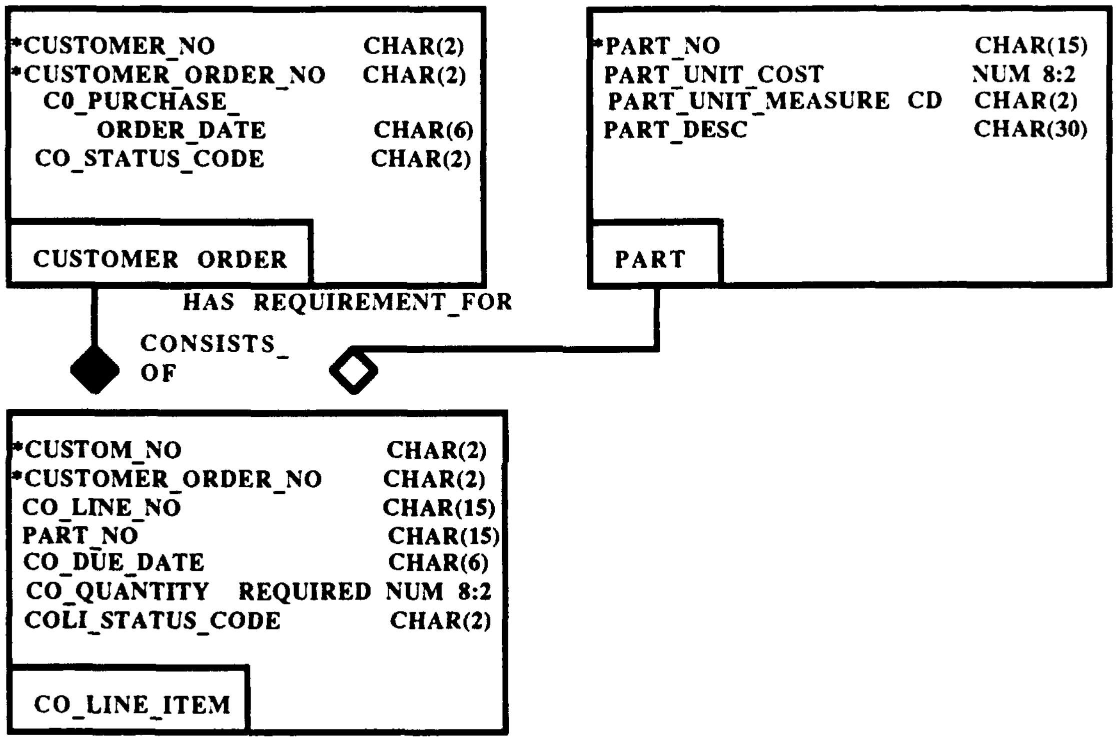

Figure C-1. Conceptual Schema IDEF1 Model

APPEND[illegible]

SAMPLE PROGRAMS

This appendix contains some sample COBOL programs. These COBOL programs are written against the USER VIEW Report provided in Appendix B.

COBOL PROGRAM 1: SINGLE-ROW SELECT

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CDSEL4.
*******************************************************
*
*  RETRIEVE THE CUSTOMER NUMBER FOR A CUSTOMER NAME
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   SHOW-CODE                PIC -----9.
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01   CUSTOMER-NAME             PIC X(40).
 01   CUSTOMER-NO               PIC 9(8).
 01   CUSTOMER-IND              PIC 9.
      EXEC SQL END DECLARE SECTION END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
      MOVE 0 TO SQLCODE.
      DISPLAY " DISPLAY THE CUSTOMER NUMBER FOR A CUSTOMER".
 ACCEPT-INPUT.
      DISPLAY " ENTER CUSTOMER NAME (QUIT TO EXIT)".
      ACCEPT CUSTOMER-NAME.
      IF CUSTOMER-NAME = "QUIT"
         GO TO EXIT-PROGRAM.
      PERFORM RETRIEVE-DATA  THRU  RETRIEVE-DATA-EXIT.
      GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
*
      EXIT PROGRAM.
*
 RETRIEVE-DATA.
         EXEC SQL
           SELECT  CUSTOMER_NO
           INTO :CUSTOMER-NO INDICATOR :CUSTOMER-IND
           FROM    CUSTOMER_ORDER
                 WHERE   CUSTOMER_NAME = :CUSTOMER-NAME
        END-EXEC.
```

```
*
     IF SQLCODE = 100
        GO TO RETRIEVE-DATA-EXIT.
     IF SQLCODE = 0
        NEXT SENTENCE
     ELSE
        MOVE SQLCODE TO SHOW-CODE
        DISPLAY "ERROR OCCURED DURING SELECT"
        DISPLAY "STATUS IS:" SHOW-CODE
        GO TO RETRIEVE-DATA-EXIT.
     DISPLAY "CUSTOMER NAME     " CUSTOMER-NAME.
     IF CUSTOMER-IND = 0
        DISPLAY "CUSTOMER NUMBER:  " CUSTOMER-NO
     ELSE
        DISPLAY "CUSTOMER NUMBER:   NULL".
 RETRIEVE-DATA-EXIT.
     EXIT.
```

COBOL PROGRAM 2: MULTI-ROW SELECT

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CDSEL2.
*******************************************************
*
*  Retrieve all EMPLOYEES for a TASK/ASSIGNMENT
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   SHOW-CODE                PIC -----9.
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01   EMPLOYEE-NO               PIC 9(8).
 01   PROJECT-NO                PIC X(15).
 01   TASK-NO                   PIC X(15).
 01   PROJECT-IND               PIC 9.
      EXEC SQL END DECLARE SECTION END-EXEC.
*
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
      MOVE 0   TO SQLCODE.
      DISPLAY " DISPLAY ALL EMPLOYEES FO:. A PROJECT/TASK".
 ACCEPT-INPUT.
      DISPLAY " ".
      DISPLAY " ENTER EMPLOYEE NUMBER (0 TO EXIT)".
      DISPLAY " ".
      ACCEPT EMPLOYEE-NO.
      IF EMPLOYEE-NO = 0
         GO TO EXIT-PROGRAM.
      PERFORM RETRIEVE-DATA  THRU  RETRIEVE-DATA-EXIT.
      GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
*
      EXIT PROGRAM.
*
*
 RETRIEVE-DATA.
         EXEC SQL DECLARE CUR1 CURSOR
           FOR   SELECT  PROJECT_NO,
                         TASK_NO
                 FROM    TASK_ASSIGNMENT
                 WHERE   EMPLOYEE_NO = :EMPLOYEE-NO
                 ORDER BY  PROJECT_NO
        END-EXEC.
*
      IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
         DISPLAY "ERROR OCCURED DURING DECLARE"
```

```
              DISPLAY "STATUS IS: " SHOW-CODE
               GO TO RETRIEVE-DATA-EXIT.
      *
           EXEC SQL OPEN CUR1
           END-EXEC.
           IF SQLCODE NOT = 0
              MOVE SQLCODE TO SHOW-CODE
              DISPLAY "ERROR OCCURED DURING OPEN CURSOR"
              DISPLAY "STATUS IS: " SHOW-CODE
               GO TO RETRIEVE-DATA-EXIT.
           PERFORM FETCH-CURSOR THRU FETCH-EXIT
               UNTIL SQLCODE NOT = 0.
           IF SQLCODE = 100
              NEXT SENTENCE
           ELSE
              MOVE SQLCODE TO SHOW-CODE
              DISPLAY "ERROR OCCURED DURING FETCH"
              DISPLAY "STATUS IS: " SHOW-CODE
              GO TO RETRIEVE-DATA-EXIT.
      *
           EXEC SQL  CLOSE CUR1 END-EXEC.
           IF SQLCODE NOT = 0
              MOVE SQLCODE TO SHOW-CODE
              DISPLAY "ERROR OCCURED DURING CLOSE CURSOR"
              DISPLAY "STATUS IS: " SHOW-CODE.
       RETRIEVE-DATA-EXIT.
           EXIT.
      *
       FETCH-CURSOR.
           EXEC SQL  FETCH CUR1 INTO :PROJECT-NO :PROJECT-IND,
                                     :TASK-NO
           END-EXEC.
           IF SQLCODE NOT = 0
              GO TO FETCH-EXIT.
           IF PROJECT-IND = 0
              DISPLAY "PROJECT NO: ", PROJECT-NO, "TASK NO: ", TASK-NO.
       FETCH-EXIT.
           EXIT.
```

COBOL PROGRAM 3: INSERT

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. INS3.
*******************************************************
*
*  Insert data into userview STAFF
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   SHOW-CODE                PIC -----9.
 01   OPTION                   PIC 9.
*
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01   HOURLY-RATE               PIC S9(12)V999.
 01   EMPLOYEE-NO               PIC 9(8).
      EXEC SQL END DECLARE SECTION END-EXEC.
*
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
      MOVE 0   TO SQLCODE.
      DISPLAY " INSERT STAFF INFORMATION".
 ACCEPT-INPUT.
      DISPLAY " ".
      DISPLAY " ENTER INPUT OPTION (1=DATA,2=QUIT)".
      ACCEPT  OPTION.
      IF OPTION = 1
         PERFORM ACCEPT-USER-INPUT THRU USER-INPUT-EXIT
         GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
      IF SQLCODE = 0
         PERFORM SQL-COMMIT THRU SQL-COMMIT-EXIT
      ELSE
         PERFORM SQL-ROLLBACK THRU SQL-ROLLBACK-EXIT.
      EXIT PROGRAM.
*
 ACCEPT-USER-INPUT.
      DISPLAY "ENTER HOURLY RATE (PIC 9(12)V999):".
      ACCEPT HOURLY-RATE.
      DISPLAY "ENTER EMPLOYEE-NO  (PIC 9(8)):".
      ACCEPT EMPLOYEE-NO.
      EXEC SQL INSERT INTO STAFF
               VALUES (:HOURLY-RATE,
                       :EMPLOYEE-NO)
      END-EXEC.
      IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
         DISPLAY "ERROR OCCURED DURING INSERT"
         DISPLAY "STATUS IS:" SHOW-CODE.
 SQL-INSERT-EXIT.
     EXIT.
*
 SQL-COMMIT.
     DISPLAY "INSERT SUCCESSFUL".
     EXEC SQL COMMIT WORK
     END-EXEC.
     IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
         DISPLAY "ERROR OCCURRED DURING COMMIT"
         DISPLAY "STATUS IS: " SHOW-CODE
 SQL-COMMIT-EXIT.
     EXIT.
*
 SQL-ROLLBACK.
     DISPLAY "INSERT BEING ROLLED BACK".
     EXEC SQL ROLLBACK WORK
     END-EXEC.
     IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
         DISPLAY "ERROR OCCURRED DURING ROLLBACK"
         DISPLAY "STATUS IS: " SHOW-CODE
 SQL-ROLLBACK-EXIT.
     EXIT.
```

COBOL PROGRAM 4: DELETE

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CDDEL3.
*
*******************************************************
*
*  DELETE data FROM  userview STAFF
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  SHOW-CODE                PIC -----9.
 01  OPTION                   PIC 9.
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  EMPLOYEE-NO               PIC 9(8).
     EXEC SQL END DECLARE SECTION END-EXEC.
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
     MOVE 0   TO SQLCODE.
 ACCEPT-INPUT.
      DISPLAY " DELETE STAFF INFORMATION".
      DISPLAY " ENTER INPUT OPTION (1=DATA,2=QUIT)".
      ACCEPT  OPTION.
      IF OPTION = 1
          PERFORM ACCEPT-USER-INPUT THRU USER-INPUT-EXIT
          GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
     IF SQLCODE NOT = 0
        EXEC SQL ROLLBACK WORK END-EXEC.
        IF SQLCODE NOT = 0
           DISPLAY "ERROR OCCURED ON ROLLBACK"
           MOVE SQLCODE TO SHOW-CODE
           DISPLAY "STATUS IS ", SHOW-CODE
        END-IF
     ELSE
        EXEC SQL COMMIT WORK END-EXEC.
        IF SQLCODE NOT = 0
           DISPLAY "ERROR OCCURED ON COMMIT"
           MOVE SQLCODE TO SHOW-CODE
           DISPLAY "STATUS IS ", SHOW-CODE.
 EXIT.
     EXIT PROGRAM.
*
 ACCEPT-USER-INPUT.
     DISPLAY "ENTER EMPLOYEE NUMBER TO BE DELETED  (PIC 9(8)):".
     ACCEPT EMPLOYEE-NO.
     EXEC SQL DELETE FROM STAFF
```

```
               WHERE EMPLOYEE_NO = :EMPLOYEE-NO
       END-EXEC.
       IF SQLCODE NOT = 0
          MOVE SQLCODE TO SHOW-CODE
          DISPLAY "ERROR OCCURED DURING DELETE "
          DISPLAY "STATUS IS:" SHOW-CODE.
   SQL-DELETE-EXIT.
       EXIT.
```

COBOL PROGRAM 5: UPDATE

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CDUPD2.
*******************************************************
*
*  UPDATE TASK_ASSIGNMENT, CHANGING EMPLOYEE HOURS
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   SHOW-CODE                PIC -----9.
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01   EMPLOYEE-NO               PIC 9(8).
 01   TASK-NO                   PIC X(15).
 01   PROJECT-NO                PIC X(15).
 01   EMPLOYEE-HOURS            PIC S9(12)V9(3).
      EXEC SQL END DECLARE SECTION END-EXEC.
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
      MOVE 0   TO SQLCODE.
      DISPLAY " CHANGING AN EMPLOYEE'S HOURS WORKED FOR A TASK.".
 ACCEPT-INPUT.
      DISPLAY " ".
      DISPLAY " ENTER EMPLOYEE NUMBER (0 TO EXIT)".
      DISPLAY " ".
      ACCEPT EMPLOYEE-NO.
      IF EMPLOYEE-NO = 0
         GO TO EXIT-PROGRAM.
      DISPLAY " ENTER NEW HOURS WORKED (PIC 9(12)V9(3)): ".
      ACCEPT EMPLOYEE-HOURS.
      DISPLAY " ENTER TASK ON WHICH EMPLOYEE WORKED: ".
      ACCEPT TASK-NO.
      DISPLAY " ENTER PROJECT ON WHICH EMPLOYEE WORKED: ".
      ACCEPT PROJECT-NO.
      PERFORM MODIFY-DATA THRU MODIFY-DATA-EXIT.
      GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
      IF SQLCODE = 0
         DISPLAY "UPDATE IS SUCCESSFUL"
         EXEC SQL COMMIT WORK END-EXEC.
         IF SQLCODE NOT = 0
            MOVE SQLCODE TO SHOW-CODE
            DISPLAY "ERROR ON COMMIT; STATUS = ", SHOW-CODE
         END-IF
      ELSE
         DISPLAY "UPDATE IS BEING ROLLED BACK"
         EXEC SQL ROLLBACK WORK END-EXEC.
         IF SQLCODE NOT = 0
            MOVE SQLCODE TO SHOW-CODE
            DISPLAY "ERROR ON ROLLBACK; STATUS = ", SHOW-CODE.
```

```
*
     EXIT PROGRAM.
*
 MODIFY-DATA.
        EXEC SQL
             UPDATE TASK_ASSIGNMENT
                SET EMPLOYEE_HOURS = :EMPLOYEE-HOURS
                WHERE TASK_NO = :TASK-NO AND
                      PROJECT_NO = :PROJECT-NO AND
                      EMPLOYEE_NO = :EMPLOYEE-NO
        END-EXEC.
     IF SQLCODE NOT = 0
        MOVE SQLCODE TO SHOW-CODE
        DISPLAY "ERROR ON UPDATE; STATUS = ", SHOW-CODE.
 MODIFY-DATA-EXIT.
      EXIT.
```

COBOL PROGRAM 6: FIOP

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. INS4.
*******************************************************
*
*  Insert data into userview STAFF
*
*******************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY ERRFS OF IISSCLIB.
 01   SHOW-CODE                PIC -----9.
 01   FIOP-STATUS              PIC X(5).
 01   STAFF-INFO.
      03  HOURLY-RATE-FILE     PIC 9(12)V999.
      03  EMPLOYEE-NO-FILE     PIC 9(8).
 01   OPTION                   PIC 9.
 01   FILE-NAME                PIC X(80).
 01   FCB-N                    PIC S9(9) COMP.
 01   REC-LEN                  PIC S9(9) COMP.
 01   NUM-OF-REC               PIC S9(9) COMP VALUE 2000.
 01   RETURN-LENGTH            PIC S9(9) COMP.
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01   HOURLY-RATE               PIC S9(12)V999.
 01   EMPLOYEE-NO               PIC 9(8).
      EXEC SQL END DECLARE SECTION END-EXEC.
 LINKAGE SECTION.
*
 PROCEDURE DIVISION.
      MOVE 0   TO SQLCODE.
      DISPLAY " INSERT STAFF INFORMATION".
 ACCEPT-INPUT.
      DISPLAY " ENTER INPUT OPTION (1=FILE,2=QUIT)".
      ACCEPT  OPTION.
      IF OPTION = 1
         PERFORM ACCEPT-FILE-INPUT THRU FILE-INPUT-EXIT
         GO TO ACCEPT-INPUT.
 EXIT-PROGRAM.
*
      EXIT PROGRAM.
*
 ACCEPT-FILE-INPUT.
      DISPLAY " ENTER FILE NAME".
      ACCEPT FILE-NAME.
      MOVE "R" TO DISPOSITION.
      CALL "OPNFIL"  USING       FCB-N
                          FIOP-STATUS
                          FILE-NAME
                          DISPOSITION
                          REC-LEN
                          NUM-OF-REC.
```

```
     IF FIOP-STATUS NOT = KES-FILE-OK
        DISPLAY "ERROR OPENING DATA FILE"
        DISPLAY "STATUS IS: " FIOP-STATUS
        GO TO FILE-INPUT-EXIT.
 NEXT-RECORD.
     MOVE SPACES TO STAFF-INFO.
     CALL "INPFIL" USING       FCB-N
                         FIOP-STATUS
                         STAFF-INFO
                         REC-LEN
                         RETURN-LENGTH.
     IF FIOP-STATUS = KES-END-OF-FILE-INPUT
        GO TO FILE-INPUT-CLOSE
     ELSE IF FIOP-STATUS NOT = KES-FILE-OK
        DISPLAY "ERROR READING DATA FILE"
        DISPLAY "STATUS IS:" RET-STATUS
        GO TO FILE-INPUT-CLOSE.
     MOVE  HOURLY-RATE-FILE  TO  HOURLY-RATE.
     MOVE  EMPLOYEE-NO-FILE  TO  EMPLOYEE-NO.
     EXEC SQL INSERT INTO STAFF
              VALUES (:HOURLY-RATE,
                      :EMPLOYEE-NO)
     END-EXEC.
     IF SQLCODE NOT = 0
        MOVE SQLCODE TO SHOW-CODE
        DISPLAY "ERROR OCCURED DURING INSERT"
        DISPLAY "STATUS IS:" SHOW-CODE
     ELSE
        GO TO NEXT-RECORD.
 FILE-INPUT-CLOSE.
     IF SQLCODE = 0
        PERFORM SQL-COMMIT THRU SQL-COMMIT-EXIT
     ELSE
        PERFORM SQL-ROLLBACK THRU SQL-ROLLBACK-EXIT.
     MOVE "K" TO DISPOSITION.
     CALL "CLSFIL" USING       FCB-N
                         FIOP-STATUS
                         DISPOSITION.
     IF FIOP-STATUS NOT = KES-FILE-OK
        DISPLAY "ERROR CLOSING DATA FILE"
        DISPLAY "STATUS IS:" FIOP-STATUS.
 FILE-INPUT-EXIT.
     EXIT.
*
 SQL-COMMIT.
     EXEC SQL COMMIT WORK END-EXEC.
     IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
         DISPLAY "ERROR OCCURRED DURING COMMIT"
         DISPLAY "STATUS IS: " SHOW-CODE.
 SQL-COMMIT-EXIT.
     EXIT.
*
 SQL-ROLLBACK.
     EXEC SQL ROLLBACK WORK END-EXEC.
     IF SQLCODE NOT = 0
         MOVE SQLCODE TO SHOW-CODE
```

```
            DISPLAY "ERROR OCCURRED DURING ROLLBACK"
            DISPLAY "STATUS IS: " SHOW-CODE.
        SQL-ROLLBACK-EXIT.
            EXIT.
```

APPENDIX E

SAMPLE C PROGRAMS

This appendix contains some sample C programs. These C programs are written against the USER VIEW Report in Appendix B.

C PROGRAM 1: SINGLE-ROW SELECT

```
/*
 *   NAME: SELC1
 ***************************************************************************/
void selc1()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char   pno[15];
   char   pno_qual[15];
   char   pdesc[50];
   double budamt;
   long   pdepno;
   char   pduedt[8];
   char   ptasno[15];
   long   pmgrno;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   int  option;
START:
   for (;;)
      {
      do
        {
        printf("\nEnter Input Option (1=DATA  2=QUIT):");
        scanf(" %d", &option);
        }
      while(option < 1 || option > 2);

      if (option == 2)
         break;
      else if (option == 1 )
         {                                /* get values from the user terminal
*/
         printf("\n Enter the Project No (up to 15 digits):");
         scanf(" %s", pno_qual);
         EXEC SQL SELECT A.PROJECT_NO,
                         B.PROJECT_DESC,
                         B.PROJ_DEPT_NO,
                         B.BUDGET_AMOUNT,
                         B.PROJECT_DUE_DATE,
                         A.TASK_NO,
                         B.MANAGER_NO
                  INTO   :pno,
                         :pdesc,
                         :pdepno,
                         :budamt,
                        :pduedt,
                        :ptasno,
                        :pmgrno
               FROM PROJECT_B B, TASK A
               WHERE B.PROJECT_NO = :pno_qual AND
                     A.PROJECT_NO  = B.PROJECT_NO (+);
         if (sqlca.sqlcode != 0)
            printf("\n SELECT FAILED WITH STATUS=%d",sqlca.sqlcode);
         else
            {
            printf("\nProject No qualified on: - %.15s", pno_qual);
            printf("\nProject No            - %.15s", pno);
            printf("\nProject Desc          - %.50s", pdesc);
            printf("\nProject Bud Amt       - %12.2f", budamt);
            printf("\nProject Due Date      - %.8s", pduedt);
            printf("\nProject Dept No.      - %d", pdepno);
            printf("\nProject Manager No.   - %d", pmgrno);
            printf("\nTask Number           - %.15s\n", ptasno);
            }
         }
      }
   return;
}
```

C PROGRAM 2: MULTI-ROW SELECT

```
/*****************************************************************************
 *   NAME:  cdselc6  -- Selects data from the view CUSTOMER_ORDER_B.
 *****************************************************************************/
void cdselc6()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char   customer_name[40];
   long order_line_no;
   long order_no;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   printf("\nSelecting from view CUSTOMER_ORDER_B: \n\n");
   EXEC SQL WHENEVER SQLERROR GOTO err;
   EXEC SQL DECLARE CUR1 CURSOR FOR
        SELECT CUSTOMER_NAME, ORDER_LINE_NO, ORDER_NO
        FROM   CUSTOMER_ORDER_B
        WHERE  ORDER_NO > 892300;
    EXEC SQL OPEN CUR1;
    while (SQLCODE != 0)
       {
       EXEC SQL FETCH CUR1 INTO :customer_name, :order_line_no, :order_no;
       printf("%.40s %d  %d\n", customer_name, order_line_no, order_no);
       }
   EXEC SQL CLOSE CUR1;
goodexit:
   return;
err:
   printf("ERROR ON SELECT FROM CUSTOMER_ORDER_B\n");
   printf("STATUS IS %d\n", sqlca.sqlcode);
   return;
}
```

C PROGRAM 3: INSERT

```
/*****************************************************************************
 *   NAME: cdinscl    Insert into PROJECT entity
 *****************************************************************************/
void inscl()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char   pno[15];
   double budamt;
   long   pdepno;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   char   tmppno[16];
   char   option[5];
   sqlca.sqlcode = 0;
   for (;;)
      {
       do
         {
          printf("\nEnter Input Option (1=DATA   2=QUIT):");
          scanf (" %s",option);
         }
       while (option[0] < '1' || option[0] > '2');

       if (option[0] == '2')
          break;
       else
         {
          strfill(pno, " ", 15);
          printf("\n Enter the Project No (up to 15 digits):");
          scanf(" %15[^\n]s",tmppno);
          strncpy(pno, tmppno, strlen(tmppno));
          printf("\n Enter the Budget Amount (10).99):");
          scanf(" %f",&budamt);
          printf("\nEnter the Project Department Number (4 integers):");
          scanf(" %d",pdepno);
         }
      EXEC SQL INSERT INTO PROJECT_B (PROJECT_NO,
                                      BUDGET_AMOUNT,
                                      PROJ_DEPT_NO)
                         VALUES (     :pno,
                                      :budamt,
                                      :pdepno);
        if (sqlca.sqlcode != 0)
           printf("\n INSERT FAILED WITH STATUS=%d",sqlca.sqlcode);
        else
           printf("\n Row successfully inserted ");
      }
   if(sqlca.sqlcode != 0)
     {
      printf("\n INSERT BEING ROLLED BACK");
      EXEC SQL ROLLBACK WORK;
      if (sqlca.sqlcode != 0)
         printf("\n ERROR ON ROLLBACK, STATUS=%d",sqlca.sqlcode);
     }
   else
```

```
        {
          printf ("\n INSERT BEING COMMITTED");
          EXEC SQL COMMIT WORK;
          if (sqlca.sqlcode != 0)
             printf("\n ERROR ON COMMIT, STATUS=%d",sqlca.sqlcode);
         }
      return;
   }
```

C PROGRAM 4: DELETE

```
/***************************************************************************fil
 *   NAME: CDDELC1 -- delete from the PROJECT_B view.
 ****************************************************************************/
void cddelc1()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char    pno[15];
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   int option;
   sqlca.sqlcode = 0;
START:
   while(option < 1 || option > 2)
     {
      printf("\nEnter Input Option (1=DEL A RECORD  2=QUIT):");
      scanf(" %d",&option);
     }
   if(option == 1)
   {
      printf("\n Enter the Project No (up to 15 digits):");
      scanf(" %15s",pno);
      EXEC SQL DELETE FROM PROJECT_B
         WHERE PROJECT_NO = :pno;
      if (sqlca.sqlcode != 0)
          printf("\n DELETE FAILED WITH STATUS=%d",sqlca.sqlcode);
      goto START;
   }
   if(option == 2)
   {
     if (sqlca.sqlcode == 0)
       {
        printf("\n DELETE BEING COMMITTED");
        EXEC SQL COMMIT WORK;
        if (sqlca.sqlcode != 0)
           printf("\n ROLLBACK FAILED WITH STATUS=%d",sqlca.sqlcode);
       }
     else
        printf("\n DELETE BEING ROLLED BACK");
        EXEC SQL ROLLBACK WORK;
        if (sqlca.sqlcode != 0)
           printf("\n COMMIT FAILED WITH STATUS=%d",sqlca.sqlcode);
       }
     return;
   }
}
```

C PROGRAM 5: UPDATE

```
/*****************************************************************************
 *   NAME: CDUPDC1 -- UPDATE the PROJECT_B view
 *****************************************************************************/
void updc1()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char   pno_qual[15];
   char   pdesc[50];
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   int option;
   for (;;)
      {
      do
         {
         printf("\nEnter Input Option (1=DATA,2=QUIT):");
         scanf(" %d", &option);
         }
      while(option < 1 || option > 2);
      if (option == 2)
         break;
      if (option == 1 )
         {
         printf("\n Enter the Project No (up to 15 characters):");
         scanf(" %15s",pno_qual);
         printf("\nEnter the new description ->");
         scanf(" %s",pdesc);
         EXEC SQL UPDATE PROJECT_B
                  SET PROJECT_DESC =     :pdesc
                  WHERE PROJECT_NO =     :pno_qual;
         if (sqlca.sqlcode != 0)
            printf("\n UPDATE FAILED WITH STATUS = %d",sqlca.sqlcode);

         }
      }
       if (sqlca.sqlcode != 0)
          {
            printf("\n UPDATE OF PROJECT_B BEING ROLLED BACK");
            EXEC SQL ROLLBACK WORK;
            if (sqlca.sqlcode != 0)
               printf("\n ROLLBACK FAILED; STATUS = %d", sqlca.sqlcode);
          }
       else
          {
            printf("\n UPDATE OF PROJECT_B SUCCESSFUL");
            EXEC SQL COMMIT WORK;
            if (sqlca.sqlcode != 0)
               printf("\n COMMIT FAILED; STATUS = %d", sqlca.sqlcode);
          }
   return;
}
```

C PROGRAM 6: FIOP

```
/*****************************************************************************
 *   NAME: insc2
 *****************************************************************************/
#include "errfs.h"
void insc2()
{
   EXEC SQL BEGIN DECLARE SECTION;
   char   pno[15];
   double budamt;
   long   pdepno;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL INCLUDE SQLCA;
   char option[5];
   int *fcb;
   char filenam[81];
   char inrec[31];
   long reclen;
   long retlen;
   char fiopst[6];
   long numorec;
   int  i;
   char workarea[15];
   void clsfil(),inpfil(),opnfil();
   sqlca.sqlcode = 0;
   fiopst[5] = '\0';
   for (;;)
      {
       do
         {
          printf("\nEnter Input Option (1=FILE    2=QUIT):");
          scanf (" %s",option);
         }
       while (option[0] < '1' || option[0] > '2');

       if (option[0] == '2')
           break;
       else if(option[0] == '1')
           {
            printf("\n What file is the data on (FILE.EXT)?");
            scanf(" %80s",filenam);
            opnfil(&fcb,fiopst,filenam,"R",&reclen,&numorec);
            if(strncmp(fiopst,SUCCESSFUL,5) != 0)
           {
            printf("\n COULD NOT OPEN FILE; STATUS = %s",fiopst);
            goto EXIT;
           }
       for (;;)
           {
            inpfil(&fcb,fiopst,inrec,&reclen,&retlen);
            if (strncmp(fiopst,END_OF_FILE_INPUT,5) == 0)
              {
               clsfil(&fcb,fiopst,"K");
               break;
              }
          else if (strncmp(fiopst,SUCCESSFUL,5) != 0)
            {
             printf("\n %s - Error in reading file\n", fiopst);
             clsfil(&fcb,fiopst,"K");
             break;
            }
          strncpy(pno,inrec,15);
          for(i = 0; i < 15; i++ ) workarea[i] = '\0';
          strncpy(workarea,inrec+15,12);
          sscanf(workarea, "%12.2f", &budamt);
          for(i = 0; i < 15; i++ ) workarea[i] = '\0';
          strncpy(workarea,inrec+27,4);
          sscanf(workarea, "%4ld", &pdepno);

      EXEC SQL INSERT INTO PROJECT_B (PROJECT_NO,
                                      BUDGET_AMOUNT,
                                      PROJ_DEPT_NO)
                         VALUES (    :pno,
                                     :budamt,
                                     :pdepno);
          if(sqlca.sqlcode != 0)
            {
             printf("\n INSERT FAILED WITH STATUS = %d",sqlca.sqlcode);
             break;
            }
          else
             printf("\n Row successfully inserted ");
        }
EXIT:
     }
   if(sqlca.sqlcode != 0)
     {
      printf ("\n INSERT BEING ROLLED BACK");
      EXEC SQL ROLLBACK WORK;
      if (sqlca.sqlcode != 0)
         printf("\n ERROR ON ROLLBACK. STATUS = %d", sqlca.sqlcode);
     }
   else
     {
       printf ("\n INSERT BEING COMMITTED");
       EXEC SQL COMMIT WORK;
       if (sqlca.sqlcode != 0)
         printf("\n ERROR ON COMMIT. STATUS = %d", sqlca.sqlcode);
     }
   return;
}
```

APPENDIX F

FORTRAN PROGRAM EXAMPLES

This appendix contains some sample FORTRAN programs. These sample FORTRAN programs are written agianst the USER VIEW Reported in Appendix B.

FORTRAN PROGRAM 1: SINGLE-ROW SELECT

```
C
C    FORTRAN PRC TO SELECT FROM THE VIEW CUSTOMER_ORDER_B.
C
      SUBROUTINE CDSELF1
      EXEC SQL BEGIN DECLARE SECTION
      CHARACTER*40   CUSNAM
      INTEGER        ORDLNO
      INTEGER        ORDNO
      EXEC SQL END DECLARE SECTION
      PRINT *, 'SELECTING FROM VIEW CUSTOMER_ORDER_B: '
      PRINT *, '------------------------------------'
      EXEC SQL WHENEVER SQLERROR GOTO 500
      EXEC SQL SELECT CUSTOMER_NAME, ORDER_LINE_NO, ORDER_NO
     *         INTO   :CUSNAM, :ORDLNO, :ORDNO
     *         FROM   CUSTOMER_ORDER_B
     *         WHERE  ORDER_NO = 5
      IF (SQLCOD .EQ. 0) THEN
         PRINT *, 'CUSTOMER_NAME: ', CUSNAM
         PRINT *, 'ORDER_LINE_NO: ', ORDLNO
         PRINT *, 'ORDER_NO: ', ORDNO
         PRINT *, '--->'
      ENDIF
      GOTO 900
  500 CONTINUE
      PRINT *,'ERROR ON SELECT FROM CUSTOMER_ORDER_B VIEW '
      PRINT *,'STATUS IS ', SQLCOD
  900 RETURN
      END
```

FORTRAN PROGRAM 2: MULTI-ROW SELECT

```
C
C   FORTRAN PRC TO SELECT FROM THE VIEW CUSTOMER_ORDER_B.
C
      SUBROUTINE CDSELF4
      EXEC SQL BEGIN DECLARE SECTION
      CHARACTER*40   CUSNAM
      INTEGER        ORDLNO
      INTEGER        ORDNO
      EXEC SQL END DECLARE SECTION
      PRINT *, 'SELECTING FROM VIEW CUSTOMER_ORDER_B: '
      PRINT *, '-------------------------------------'
      EXEC SQL WHENEVER SQLERROR GOTO 900
      EXEC SQL DECLARE CUR1 CURSOR FOR
     *         SELECT CUSTOMER_NAME, ORDER_ INE_NO, ORDER_NO
     *         FROM   CUSTOMER_ORDER_B
     *         WHERE  ORDER_NO > 5
     *         ORDER BY ORDER_LINE_NO, ORDER_NO DESC
      EXEC SQL OPEN CUR1
  300 EXEC SQL FETCH CUR1 INTO :CUSNAM, :ORDLNO, :ORDNO
      IF (SQLCOD .EQ. 0) THEN
         PRINT *, 'CUSTOMER_NAME: ', CUSNAM
         PRINT *, 'ORDER_LINE_NO: ', ORDLNO
         PRINT *, 'ORDER_NO: ', ORDNO
         PRINT *, '--->'
         GO TO 300
      ENDIF
      EXEC SQL CLOSE CUR1
      GOTO 1000
  900 CONTINUE
      PRINT *,'ERROR ON SELECT FROM CUSTOMER_ORDER_B VIEW '
      PRINT *,'STATUS = ', SQLCOD
 1000 RETURN
      END
```

FORTRAN PROGRAM 3: INSERT

```
      SUBROUTINE INSF1
      INTEGER OPTION
      EXEC SQL BEGIN DECLARE SECTION
      INTEGER EMPNO
      DOUBLE PRECISION SALARY
      EXEC SQL END DECLARE SECTION
      EXEC SQL INCLUDE SQLCA
      SQLCOD = 0
  100 CONTINUE
      PRINT *,'ENTER INPUT OPTION (1=DATA,2=QUIT):'
      READ(5,1010)OPTION
 1010 FORMAT(I1)
C
      IF (OPTION.EQ.1) GOTO 200
      IF (OPTION.EQ.2) THEN
        IF (SQLCOD .EQ. 0) THEN
           PRINT *, 'INSERT SUCCESSFUL'
           EXEC SQL COMMIT WORK
           IF (SQLCOD .NE. 0)
     -        PRINT *,'COMMIT UNSUCCESSFUL; STATUS = ',SQLCOD
        ELSE
           PRINT *, 'INSERT BEING ROLLED BACK'
           EXEC SQL ROLLBACK WORK
           IF (SQLCOD .NE. 0)
     -        PRINT *,'ROLLBACK UNSUCCESSFUL; STATUS = ',SQLCOD
        ENDIF
        RETURN
      ENDIF
      GOTO 100
  200 CONTINUE
      PRINT *,' ENTER THE MANAGER EMPLOYEE NUMBER (UP TO 8 DIGITS):'
      READ(5,1030)EMPNO
 1030 FORMAT(I8)
      PRINT *,' ENTER THE SALARY FOR EMPLOYEE ',EMPNO,' (9.2):'
      READ(5,1050)SALARY
 1050 FORMAT(F9.2)
      EXEC SQL INSERT INTO MANAGER_B
     *                   (EMPLOYEE_NO, MANAGER_SALARY)
     *             VALUES (:EMPNO,     :SALARY)
      IF (SQLCOD.NE.0) PRINT *, 'INSERT UNSUCCESSFUL, STATUS = ', SQLCOD
      GOTO 100
      END
```

FORTRAN PROGRAM 4: DELETE

```
      SUBROUTINE CDDELF1
      INTEGER        OPTION
      EXEC SQL BEGIN DECLARE SECTION
      INTEGER EMPNO
      DOUBLE PRECISION  EMPSAL
      EXEC SQL END DECLARE SECTION
      EXEC SQL INCLUDE SQLCA
 100  CONTINUE
      PRINT *,'ENTER INPUT OPTION (1=DATA,2=QUIT)'
      READ(5,1010)OPTION
1010  FORMAT(I1)
      IF (OPTION .LT. 1 .OR. OPTION .GT. 2) GOTO 100
      IF (OPTION .EQ. 2) GOTO 500
      PRINT *,' ENTER THE EMPLOYEE NUMBER:'
      READ(5,1020)EMPNO
1020      FORMAT(I8)
      EXEC SQL DELETE FROM MANAGER_B WHERE EMPLOYEE_NO = :EMPNO
      IF (SQLCOD.NE.0)
     -   PRINT *, 'ERROR ON DELETE.  STATUS IS ',SQLCOD
      GOTO 100
 500  CONTINUE
      IF (SQLCOD .EQ. 0) THEN
         EXEC SQL COMMIT WORK
         IF (SQLCOD .EQ. 0) THEN
            PRINT *,' DELETES PERFORMED AND COMMITTED'
         ELSE
            PRINT *,' ERROR ON COMMIT.  STATUS IS ', SQLCOD
         ENDIF
      ELSE
         EXEC SQL ROLLBACK WORK
         IF (SQLCOD .EQ. 0) THEN
            PRINT *,' DELETES ROLLED BACK'
         ELSE
            PRINT *,' ERROR ON ROLLBACK.  STATUS IS ', SQLCOD
         ENDIF
      ENDIF
      RETURN
      END
```

FORTRAN PROGRAM 5: UPDATE

```
      SUBROUTINE CDUPDF2
C
C   FORTRAN PRC TO MODIFY DATA IN THE ORDER_LINE_ITEM_B VIEW
C
      INTEGER OPTION
      EXEC SQL BEGIN DECLARE SECTION
      INTEGER             QCUST
      INTEGER             MQUANT
      EXEC SQL END DECLARE SECTION
      PRINT *, 'MODIFY PART NUMBER AND QUANTITY FOR A '
      PRINT *, 'LINE ITEM OF A CUSTOMER ORDER'
  100 CONTINUE
      PRINT *, 'ENTER CUSTOMER NUMBER (0 TO QUIT) '
      READ (5,1010)QCUST
 1010 FORMAT(I8)
      IF (CUSTNO .EQ. 0)  GOTO 900
      PRINT *, 'ENTER NEW QUANTITY (8 DIGITS)'
      READ (5,1050)MQUANT
 1050 FORMAT(I8)
      EXEC SQL UPDATE ORDER_LINE_ITEM_B
     1           SET QUANTITY = :MQUANT
     1          WHERE CUSTOMER_NO = :QCUST
      IF (SQLCOD .NE. 0) PRINT *, 'ERROR ON UPDATE; STATUS = ',SQLCOD
      GOTO 100
  900 IF (SQLCOD .EQ. 0) THEN
         PRINT *, 'UPDATE SUCCESSFUL'
         EXEC SQL COMMIT WORK
         IF (SQLCOD .NE. 0)
     -      PRINT *, 'ERROR ON COMMIT; STATUS = ', SQLCOD
      ELSE
         PRINT *, 'UPDATE BEING ROLLED BACK'
         EXEC SQL ROLLBACK WORK
         IF (SQLCOD .NE. 0)
     -      PRINT *, 'ERROR ON ROLLBACK; STATUS = ', SQLCOD
      ENDIF
      RETURN
      END
```

FORTRAN PROGRAM 6: FIOP

```
      SUBROUTINE INSF2
$ALIAS OPNFIL C (%REF, %REF, %REF, %REF, %REF, %REF)
$ALIAS CLSFIL C (%REF, %REF, %REF)
$ALIAS INPFIL C (%REF, %REF, %REF, %REF, %REF)
       INTEGER          FCB,INLEN,RETLEN
       CHARACTER*20     INREC
       CHARACTER*80     FILNAM
       CHARACTER*1      ACCMDE
       CHARACTER*5      FIOPST
       INTEGER          RC
       INTEGER          FILOPN
       INTEGER          RECLEN
       INTEGER          OPTION
       EXEC SQL BEGIN DECLARE SECTION
       INTEGER EMPNO
       DOUBLE PRECISION SALARY
       EXEC SQL END DECLARE SECTION
       EXEC SQL INCLUDE SQLCA
       SQLCOD = 0
       FILOPN = 0
   100 CONTINUE
       PRINT *,'ENTER INPUT OPTION (1=FILE,2=QUIT):'
       READ(5,1010)OPTION
  1010 FORMAT(I1)
       IF (OPTION .LT. 1 OR OPTION .GT. 2) GOTO 100
       IF (OPTION .EQ. 2) GOTO 900
       PRINT *,' WHAT FILE IS THE DATA ON (FILE.EXT)?'
       READ (5,3020)FILNAM
  3020 FORMAT(A80)
       ACCMDE='R'
       CALL OPNFIL(FCB,FIOPST,FILNAM,ACCMDE,RECLEN,NUMREC)
       IF(FIOPST .NE. '00000') THEN
          PRINT *,' COULD NOT OPEN FILE; STATUS = ',FIOPST
          GOTO 100
       ELSE
          FILOPN = 1
       ENDIF
   400 CONTINUE
       CALL INPFIL(FCB,FIOPST,INREC,INLEN,RETLEN)
       IF (FIOPST .EQ. '11210') GOTO 500
       IF (FIOPST .NE. '00000') THEN
          PRINT *, 'ERROR ON FILE READ; STATUS = ',FIOPST
          GOTO 500
       ENDIF
       CALL CHRINT(INREC(1:4),EMPNO,RC)
       IF (RC .NE. 0) THEN
          PRINT *, 'ERROR ON CONVERSION OF EMPNO'
          GOTO 500
       ENDIF
       CALL CHDP(INREC(9:20),SALARY,RC)
       IF (RC .NE. 0) THEN
          PRINT *, 'ERROR ON CONVERSION OF SALARY'
          GOTO 500
       ENDIF
       SALARY = SALARY / 100.
      EXEC SQL INSERT INTO MANAGER_B
     *                       (EMPLOYEE_NO, MANAGER_SALARY)
     *                 VALUES (:EMPNO,      :SALARY)
      IF (SQLCOD .NE. 0) THEN
         PRINT *, 'ERROR ON INSERT; STATUS = ', SQLCOD
         GOTO 500
      ENDIF
      GOTO 400
  500 ACCMDE = 'K'
      CALL CLSFIL (FCB,FIOPST,ACCMDE)
      IF (FIOPST .NE. '00000')
     -   PRINT *, 'ERROR ON FILE CLOSE; STATUS = ',FIOPST
      GOTO 100
  900 CONTINUE
      IF (SQLCOD .NE. 0) THEN
         PRINT *,'ERROR ON INSERT TO MANAGER_B; STATUS = ', SQLCOD
         EXEC SQL ROLLBACK WORK
         IF (SQLCOD .NE. 0)
     -      PRINT *,'ERROR ON ROLLBACK; STATUS = ', SQLCOD
      ELSE
         EXEC SQL COMMIT WORK
         IF (SQLCOD .NE. 0)
     -      PRINT *, 'ERROR ON COMMIT; STATUS = ', SQLCOD
      ENDIF
      RETURN
      END
```

APPENDIX G

DATE COMPLEX MAPPING ALGORITHMS

Complex mapping algorithms have been written to support the user in accessing database columns with date data. Each algorithm has a two-fold functionality. It can receive a field in a DBMS specific format, and output a field in CDM specific format, or vice-versa. The CDM specific format referred to is YYYYMMDD. There is a algorithm for each of the DBMS's supported by the CDM, written in each language that may be generated to access the particular DBMS. The algorithms, the DBMS accessed, and the language written in are:

Algorithm	DBMS	Language
DRACOB.COB	ORACLE	COBOL
ORACC.C	ORACLE	C
ORAFOR.FOR	ORACEL	FORTRAN
INGCOB.COB	INGRES5, INGRES6	COBOL
INGC.C	INGRES5, INGRES6	C
INGFOR.FOR	INGRES5, INGRES6	FORTRAN
DB2COB.COB	DB2	COBOL
DB2C.C	DB2	C
DB2FOR.FOR	DB2	FORTRAN
VAXCOB.COB	VAX-11	COBOL

In changing the appropriate algorithm the user should know which DBMS is being accessed as well as the request processor generated language being used in accessing the particular date database column.

For example, if the DBMS type to be mapped to was INGRES5, and request processors were going to be generated in FORTRAN, the algorithm to be used should be INGFOR.FOR. If the language for the algorithm is different than the language of request processors accessing the column, unexpected results may occur.

Once the appropriate algorithm is determined, it should be defined tc the CDM using the NDDL. The file DATECMA.DAT has been supplied to assist in this process. The file will need to be customized and run for each complex mapping algorithm used. The sections DEFINE MODULE and DEFINE ALGORITHM should be read in the NDDL Reference Manual before attempting this customization. Note that for a particular algorithm, the DEFINE MODULE needs to be run only once, but the DEFINE ALGORITHM command needs to be run twice for each tag to datafield mapping used, to define the algorithm for both update and retrieval.

The following editing steps need to be performed on the file DATECMA.DAT.

1. Substitute the username and password of your CDM for UN and PW, in the first line.

2. Substitute your module name for the word "file" in each command. Do not include the file extension.

3. Substitute the language of the module for the word "language" in the DEFINE MODULE command.

4. Substitute a previously defined datatype for each occurence of the word "datatype" in the DEFINE MODULE command.

a. The type and size of the datatype for the CDM-DATE parameter needs to be CHARACTER or VARCHAR with a size of 8.

b. The type of the datatype for the DBMS-DATE parameter needs to be CHARACTER or VARCHAR. The size of the datatype is dependent on the particular DBMS. The appropriate DBMS and sizes are as follows:

DBMS	Size
ORACLE	9
INGRES5	11
INGRES6	11
VAX-11	23
DB2	6

c. The type and size of the datatype for the DIRECTION parameter needs to be CHARACTER or VARCHAR with a size of 1.

d. The datatype of the RET-STATUS parameter uses a standard CDM datatype of RET-STATUS. This does not need to be altered in the file.

5. Substitute the entity name and tag name to be mapped for "ecname.tagname" in both DEFINE ALGORITHM commands. The type and size of the tag needs to be CHARACTER or VARCHAR with a size of 8.

6. Substitute the database name, record name, and datafield name to be mapped for "dbname.recordname.fieldname" in both DEFINE ALGORITHM commands. The type of the datafield needs to be CHARACTER or VARCHAR with the size the same as determined in step 4.b.

7. For each tag to datafield mapping using the date algorithm, the two DEFINE ALGORITHM statements need to be duplicated with the appropriate changes made for "ecname.tagname" and "dbname.recordname.fieldname." The MOD INSTANCE needs to be increased each time the DEFINE ALGORITHM command is used for a particular module.

All the algorithms are compiled and included in a CDM supplied library that will be linked in appropriately by GENAP. These algorithms were written to be used when mapping a conceptual schema tag to an internal schema datafield. The external schema dataitem which is mapped to the tag using this algorithm should be defined with the same type and size as the tag.

To change the format of the dataitem from the standard CDM date format, another complex mapping can be written by the user. However, if the standard date format is altered, any ORDER BY in a SELECT statement on the dataitem is not advised, unless the following steps be performed:

1. Define the type and size of the external schema dataitem to be the same as the conceptual schema tag it is mapped to.

2. Do not use a complex mapping algorithm to map the dataitem to the tag.

3. Perform the logic to do the desired format conversion in the users application after the SELECT or FETCH statements, but before displaying or using the retrieved value.

```
    UN PW
    /*  Define the CMA module to the CDM.
    */

    DEFINE MODULE file IN language
     PARAMETERS  CDM-DATE     TYPE  datatype      /*  character 8  */
                 DBMS-DATE    TYPE  datatype
                 DIRECTIONS   TYPE  datatype      /*  character 1  */
                 RET-STATUS   TYPE  RET-STATUS /*  character 5  */
;

/*  Define the mappings from Conceptual Schema to Internal Schema.
*/

DEFINE ALGORITHM file 1 FOR UPDATE FOR PREFERENCE 1
USING PARAMETERS
CDM DATE      FROM ATTRIBUTE   ecname.tagname
DBMS-DATE     TO DATAFIELD     dbname.recordname.fieldname
DIRECTIONS    CONSTANT         'U'
STATUS;

/*  Define the mappings from Internal Schema to Conceptual Schema.
*/

DEFINE ALGORITHM file 2 FOR RETRIEVAL FOR PREFERENCE 1
USING PARAMETERS
CDM-DATE      TO ATTRIBUTE     ecname.tagname
DBMS-DATE     FROM DATAFIELD   dbname.recordname.fieldname
DIRECTIONS    CONSTANT         'R'
STATUS:

HALT;
```