

**CA-IDMS®**

---

SQL Programming  
15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

**Second Edition, October 2001**

© 2001 Computer Associates International, Inc.  
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# Contents

---

<b>How to Use This Manual</b>	ix
<b>Chapter 1. SQL Application Development in CA-IDMS</b>	1-1
1.1 About this chapter	1-3
1.2 Accessing data using SQL	1-4
1.2.1 SQL data access	1-4
1.2.2 Integrity constraints	1-5
1.3 Accessing non-SQL defined databases	1-8
1.4 SQL application development	1-10
1.4.1 Writing the application	1-10
1.4.2 Creating executable modules	1-10
1.4.3 Executing the application	1-12
1.4.4 Testing and debugging the application	1-13
<b>Chapter 2. Writing an SQL Program</b>	2-1
2.1 About this chapter	2-3
2.2 Host variables	2-4
2.2.1 SQL declare sections	2-5
2.2.2 INCLUDE TABLE directive	2-5
2.2.3 Referring to host variables	2-7
2.3 SQL sessions	2-8
2.3.1 Beginning and ending an SQL session	2-8
2.3.2 Session management options	2-9
2.4 SQL transactions	2-10
2.4.1 Beginning and ending an SQL transaction	2-10
2.5 Effect of teleprocessing statements	2-11
2.6 Concurrency control and isolation levels	2-13
2.7 SQL status-checking and error-handling	2-15
2.7.1 The SQLCA	2-15
2.7.2 Displaying SQL Communication Area fields	2-20
2.7.3 Error handling	2-21
2.7.4 Checking specific errors	2-21
<b>Chapter 3. Data Manipulation with SQL</b>	3-1
3.1 About this chapter	3-3
3.2 Data manipulation operations	3-4
3.2.1 Retrieving data	3-4
3.2.2 Adding data	3-6
3.2.3 Modifying data	3-7
3.2.4 Deleting data	3-9
3.2.5 Using indicator variables in data manipulation	3-10
3.3 Using a cursor	3-12
3.3.1 Declaring a cursor	3-12
3.3.2 Fetching a row	3-13
3.3.3 Executing a positioned update or delete	3-15
3.4 Bulk processing	3-19
3.4.1 Executing a bulk fetch	3-19

3.4.2 Executing a bulk select . . . . .	3-22
3.4.3 Executing a bulk insert . . . . .	3-23
3.5 Invoking external routines . . . . .	3-25
3.5.1 Call Statement . . . . .	3-25
3.5.1.1 CALL of a procedure . . . . .	3-25
3.5.1.2 CALL of a table procedure . . . . .	3-25
3.5.2 SELECT statement . . . . .	3-26
3.5.2.1 SELECT of a procedure . . . . .	3-26
3.5.2.2 SELECT of a table procedure . . . . .	3-26
<b>Chapter 4. Requirements and Options for Host Languages . . . . .</b>	<b>4-1</b>
4.1 About this chapter . . . . .	4-3
4.2 Using SQL in a CA-ADS application . . . . .	4-4
4.2.1 Embedding SQL statements . . . . .	4-4
4.2.1.1 Delimited, continued, and commented statements . . . . .	4-4
4.2.1.2 Placing an SQL statement . . . . .	4-5
4.2.2 Defining host variables . . . . .	4-6
4.2.2.1 Including tables . . . . .	4-8
4.2.2.2 Defining bulk structures . . . . .	4-9
4.2.3 Referring to host variables . . . . .	4-10
4.2.4 Including SQL Communication Areas . . . . .	4-10
4.3 Using SQL in a COBOL application program . . . . .	4-12
4.3.1 Embedding SQL statements . . . . .	4-12
4.3.1.1 Delimited, continued, and commented statements . . . . .	4-12
4.3.1.2 Placing an SQL statement . . . . .	4-13
4.3.2 Defining host variables . . . . .	4-14
4.3.2.1 Using COBOL data declarations . . . . .	4-14
4.3.2.2 Using INCLUDE TABLE . . . . .	4-17
4.3.2.3 Defining bulk structures . . . . .	4-19
4.3.2.4 Non-bulk structures and indicator arrays . . . . .	4-20
4.3.3 Referring to host variables . . . . .	4-22
4.3.4 Including SQL Communication Areas . . . . .	4-23
4.3.5 Copying information from the dictionary . . . . .	4-24
4.3.6 COPY IDMS FILE statement . . . . .	4-25
4.3.6.1 Purpose . . . . .	4-25
4.3.6.2 Syntax . . . . .	4-25
4.3.6.3 Parameters . . . . .	4-25
4.3.6.4 Usage . . . . .	4-25
4.3.7 COPY IDMS RECORD statement . . . . .	4-25
4.3.7.1 Purpose . . . . .	4-25
4.3.7.2 Syntax . . . . .	4-26
4.3.7.3 Parameters . . . . .	4-26
4.3.7.4 Usage . . . . .	4-26
4.3.8 COPY IDMS MODULE statement . . . . .	4-27
4.3.8.1 Purpose . . . . .	4-27
4.3.8.2 Syntax . . . . .	4-27
4.3.8.3 Parameters . . . . .	4-27
4.3.8.4 Usage . . . . .	4-27
4.3.9 INCLUDE module-name statement . . . . .	4-28
4.3.10 Non-SQL precompiler directives . . . . .	4-28
4.4 Using SQL in a PL/I application program . . . . .	4-29

4.4.1	Embedding SQL statements . . . . .	4-29
4.4.1.1	Declaring SQLXQ1 . . . . .	4-29
4.4.1.2	Delimited, continued, and commented statements . . . . .	4-29
4.4.2	Defining host variables . . . . .	4-30
4.4.2.1	Using PL/I declarations . . . . .	4-30
4.4.2.2	Using INCLUDE TABLE . . . . .	4-32
4.4.2.3	Defining bulk structures . . . . .	4-34
4.4.3	Referring to host variables . . . . .	4-35
4.4.4	Including SQL Communication Areas . . . . .	4-36
4.4.5	Including information from the dictionary . . . . .	4-37
4.4.6	INCLUDE IDMS record statement . . . . .	4-38
4.4.6.1	Purpose . . . . .	4-38
4.4.6.2	Syntax . . . . .	4-38
4.4.6.3	Parameters . . . . .	4-38
4.4.6.4	Usage . . . . .	4-39
4.4.7	INCLUDE IDMS MODULE statement . . . . .	4-39
4.4.7.1	Purpose . . . . .	4-39
4.4.7.2	Syntax . . . . .	4-39
4.4.7.3	Parameters . . . . .	4-39
4.4.8	INCLUDE module-name statement . . . . .	4-40
4.4.9	Non-SQL precompiler directives . . . . .	4-40
<b>Chapter 5. Preparing and Executing the Program . . . . .</b>		<b>5-1</b>
5.1	About this chapter . . . . .	5-3
5.2	Precompiling the program . . . . .	5-4
5.2.1	About the precompiler . . . . .	5-4
5.2.2	Precompiler options . . . . .	5-5
5.2.2.1	Syntax . . . . .	5-5
5.2.2.2	Parameters . . . . .	5-6
5.3	Compiling the program . . . . .	5-10
5.4	Creating the access module . . . . .	5-11
5.4.1	Overriding access module defaults . . . . .	5-11
5.4.2	Altering an access module . . . . .	5-15
5.5	Executing the application . . . . .	5-16
5.6	Testing the access module . . . . .	5-17
5.7	Debugging the application . . . . .	5-18
5.7.1	The Command Facility . . . . .	5-18
5.7.2	SQL trace facility . . . . .	5-19
5.7.3	EXPLAIN statement . . . . .	5-19
5.7.4	Online debugger . . . . .	5-20
<b>Chapter 6. SQL Programming Techniques . . . . .</b>		<b>6-1</b>
6.1	About this chapter . . . . .	6-3
6.2	Modularized programming . . . . .	6-4
6.2.1	Sharing a cursor . . . . .	6-4
6.2.2	Using the SET ACCESS MODULE statement . . . . .	6-7
6.3	Pseudoconversational programming . . . . .	6-9
6.3.1	Using SUSPEND SESSION and RESUME SESSION . . . . .	6-9
6.3.2	Scrolling through a list of rows . . . . .	6-10
6.3.3	Updating a row after a pseudoconverse . . . . .	6-11

6.4 Managing concurrent sessions . . . . .	6-15
6.4.1 Session management concepts . . . . .	6-15
6.4.2 Implementing concurrent sessions . . . . .	6-16
6.5 Creating and using a temporary table . . . . .	6-18
6.6 Bill-of-materials explosion . . . . .	6-21
6.6.1 What to do . . . . .	6-21
6.6.2 Sample program . . . . .	6-24
<b>Chapter 7. Using Dynamic SQL . . . . .</b>	<b>7-1</b>
7.1 About this chapter . . . . .	7-3
7.2 About dynamic SQL . . . . .	7-4
7.3 Dynamic insert, update, and delete operations . . . . .	7-6
7.3.1 Using EXECUTE IMMEDIATE . . . . .	7-6
7.3.2 Using PREPARE . . . . .	7-7
7.3.3 Using EXECUTE . . . . .	7-9
7.4 Executing prepared SELECT statements . . . . .	7-10
7.4.1 What to do . . . . .	7-10
7.4.2 Sample program . . . . .	7-11
7.5 Executing prepared CALL statements . . . . .	7-16
7.5.1 What to do . . . . .	7-16
7.5.2 Sample program . . . . .	7-18
<b>Appendix A. Sample JCL . . . . .</b>	<b>A-1</b>
A.1 About this appendix . . . . .	A-3
A.2 OS/390 . . . . .	A-4
A.3 VSE/ESA . . . . .	A-10
A.3.1 Usage . . . . .	A-12
A.4 VM/ESA . . . . .	A-14
A.4.1 Usage . . . . .	A-16
A.5 BS2000/OSD . . . . .	A-19
<b>Appendix B. Test Database . . . . .</b>	<b>B-1</b>
B.1 About this appendix . . . . .	B-3
B.2 Table names and descriptions . . . . .	B-4
B.2.1 ASSIGNMENT . . . . .	B-4
B.2.2 BENEFITS . . . . .	B-5
B.2.3 CONSULTANT . . . . .	B-6
B.2.4 COVERAGE . . . . .	B-6
B.2.5 DEPARTMENT . . . . .	B-6
B.2.6 DIVISION . . . . .	B-7
B.2.7 EMPLOYEE . . . . .	B-7
B.2.8 EXPERTISE . . . . .	B-7
B.2.9 INSURANCE_PLAN . . . . .	B-8
B.2.10 JOB . . . . .	B-8
B.2.11 POSITION . . . . .	B-9
B.2.12 PROJECT . . . . .	B-9
B.2.13 SKILL . . . . .	B-9
B.3 Test data . . . . .	B-10
B.3.1 Departments . . . . .	B-10
B.3.2 Divisions . . . . .	B-10
B.3.3 Insurance plans . . . . .	B-11

B.3.4 Jobs . . . . .	B-11
B.3.5 Projects . . . . .	B-12
B.3.6 Skills . . . . .	B-12
B.4 Test database DDL . . . . .	B-14
B.5 Demo Data . . . . .	B-22
<b>Appendix C. Precompiler Directives . . . . .</b>	<b>C-1</b>
C.1 About this appendix . . . . .	C-3
C.2 Overriding DDLDML area ready mode . . . . .	C-4
C.2.1 Syntax . . . . .	C-4
C.2.2 Parameters . . . . .	C-4
C.3 No logging of program activity statistics . . . . .	C-5
C.3.1 Syntax . . . . .	C-5
C.3.2 Parameters . . . . .	C-5
C.4 Generating a source listing . . . . .	C-6
C.4.1 Syntax . . . . .	C-6
C.4.2 Parameters . . . . .	C-6
C.5 Usage . . . . .	C-7
<b>Appendix D. Calls to IDMSIN01 . . . . .</b>	<b>D-1</b>
D.1 About this appendix . . . . .	D-3
D.2 How to call IDMSIN01 . . . . .	D-4
D.3 Example . . . . .	D-5
<b>Index . . . . .</b>	<b>X-1</b>



## **How to Use This Manual**

---

## **What this manual is about**

This manual is a guide for CA-IDMS users who are responsible for developing application programs using embedded SQL. It also documents aspects of CA-IDMS that are specific to application programming with SQL, including precompiler options and data type conversions between the database and the program language.

## Who should use this manual

This manual is for CA-IDMS application designers and programmers.

Users of this manual should be experienced in using the program language and should have a working knowledge of SQL. Users should also be familiar with concepts of CA-IDMS.

For users new to SQL, completion of the *CA-IDMS SQL Self-Training Guide* is recommended before using this manual. For further information, see “Related documentation” on page xiv at the end of this preface.

# How this manual is organized

## How the topics are grouped

- **Chapter 1** gives a brief overview of CA-IDMS, steps in SQL application development, and accessing data with SQL.
- **Chapters 2 and 3** present information about writing an application program with embedded SQL statements. Chapter 2 presents the general requirements for writing a program with embedded SQL, and Chapter 3 contains information specifically about data manipulation with SQL in the CA-IDMS environment.
- **Chapter 4** presents information about SQL programming that is specific to each programming language.
- **Chapter 5** explains the procedure for precompiling the program and creating an access module. It also presents information about executing the program and about debugging the program.
- **Chapter 6** presents various programming techniques specific to accessing CA-IDMS with SQL, including pseudoconversational programming and a bill-of-materials example.
- **Chapter 7** explains what dynamic SQL is and how you can use dynamic SQL in a program.
- The **appendices** contain:
  - Sample JCL
  - Information about the installation test database
  - Precompiler directives
  - CA-IDMS functions available through calls to IDMSIN01

**How examples are presented in this manual:** All program examples in this manual are in COBOL unless otherwise indicated.

Most examples of access to an SQL-defined database refer to a test database of employee information that is supplied as part of CA-IDMS installation. Partial documentation of this database appears in Appendix B, “Test Database.”

## How product names are referenced

This manual uses the term CA-IDMS to refer to any one of the following CA-IDMS components:

- CA-IDMS/DB — The database management system
- CA-IDMS/DC — The data communications system and proprietary teleprocessing monitor
- CA-IDMS/UCF — The universal communications facility for accessing CA-IDMS database and data communications services through another teleprocessing monitor, such as CICS
- CA-IDMS/DDS — The distributed database system

This manual uses the terms DB, DC, UCF, DC/UCF, and DDS to identify the specific CA-IDMS component only when it is important to your understanding of the product.

## Related documentation

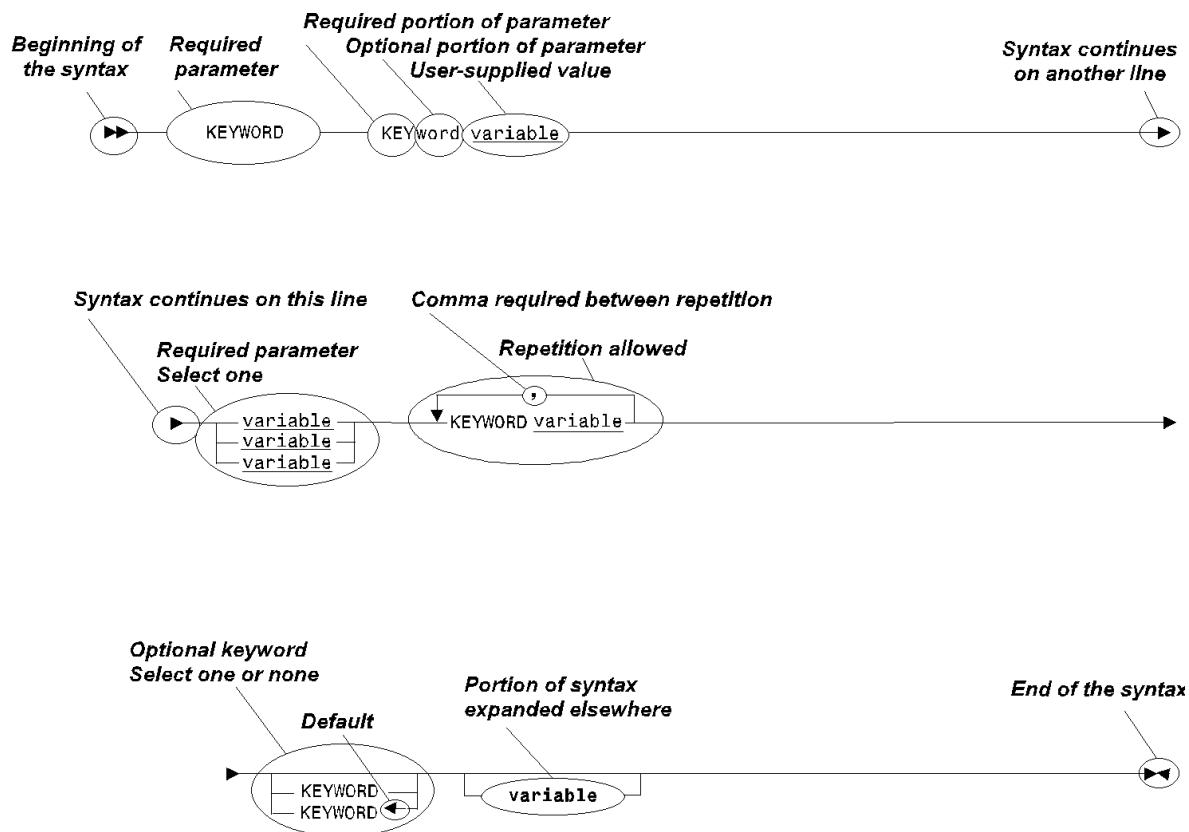
- *CA-IDMS SQL Reference Guide*
- *CA-IDMS Command Facility*
- *CA-ADS Reference Guide*
- *CA-IDMS DML Reference - COBOL*
- *CA-IDMS DML Reference - PL/I*
- *CA-IDMS Navigational DML Programming*
- *CA-IDMS Messages and Codes*
- *CA-IDMS SQL Self-Training Guide*

# Understanding syntax diagrams

Look at the list of notation conventions below to see how syntax is presented in this manual. The example following the list shows how the conventions are used.

UPPERCASE OR SPECIAL CHARACTERS	Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.
lowercase	Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.
<u>underlined lowercase</u>	Represents a value that you supply.
←	Points to the default in a list of choices.
<b>lowercase bold</b>	Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.
►—————	Shows the beginning of a complete piece of syntax.
—————►	Shows the end of a complete piece of syntax.
—————→	Shows that the syntax continues on the next line.
—————►	Shows that the syntax continues on this line.
—————→	Shows that the parameter continues on the next line.
—————→	Shows that a parameter continues on this line.
►————— parameter —————→	Shows a required parameter.
► [ parameter ] —————→	Shows a choice of required parameters. You must select one.
► [ parameter ] —————→	Shows an optional parameter.
► [ parameter ] —————→	Shows a choice of optional parameters. Select one or none.
►—[ parameter ]————→	Shows that you can repeat the parameter or specify more than one parameter.
►—[ parameter ]————→	Shows that you must enter a comma between repetitions of the parameter.

## Sample syntax diagram



# Chapter 1. SQL Application Development in CA-IDMS

---

1.1 About this chapter . . . . .	1-3
1.2 Accessing data using SQL . . . . .	1-4
1.2.1 SQL data access . . . . .	1-4
1.2.2 Integrity constraints . . . . .	1-5
1.3 Accessing non-SQL defined databases . . . . .	1-8
1.4 SQL application development . . . . .	1-10
1.4.1 Writing the application . . . . .	1-10
1.4.2 Creating executable modules . . . . .	1-10
1.4.3 Executing the application . . . . .	1-12
1.4.4 Testing and debugging the application . . . . .	1-13



## 1.1 About this chapter

This chapter is an overview of how an application program accesses data using SQL and what's involved in developing such a program. It introduces concepts and terminology that are important to understanding the discussion in the following chapters.

## 1.2 Accessing data using SQL

You embed SQL statements in an application program to access the database. SQL allows you to access the database without reference to its physical characteristics.

A database defined with SQL DDL includes constraints that govern data manipulation. The DBMS enforces constraints at runtime.

### 1.2.1 SQL data access

**Tables and views:** Data accessed through SQL is perceived as tables made up of rows and columns. A table is a **base** table.

An application program accesses an SQL-defined database by issuing SQL statements that refer to one or more base tables, or to a predefined view of one or more base tables.

**Schema and area:** A **schema** is a named collection of tables and views. The rows of a table are stored in the **area** that is specified in the CREATE TABLE statement or, if not specified, in the default area for the schema.

Concurrent access to data can be controlled at the area level and the table row level.

**SELECT statement:** A SELECT statement requests the DBMS to retrieve data. The table of values returned to the program on a select is a **result** table. Typically, a result table is a subset of the row and column values in one or more base tables.

**Cursor:** A cursor is an SQL programming construct that is used to process data in a result table. The cursor defines the result table, and the program can retrieve each row of the result table one at a time with a FETCH statement.

The cursor row whose values are available to the program represents the **cursor position**. Each FETCH statement advances the cursor position to the next row of the result table.

**Updateable cursor:** If the cursor definition meets certain requirements, it is an **updateable** cursor. The program can update or delete the row on which an updateable cursor is positioned, (that is, the row most recently fetched).

**INSERT, UPDATE, and DELETE:** The SQL statement to add a row to a table is INSERT and to delete a row is DELETE. The statement to modify one or more column values in a row is UPDATE.

**Host variables:** A host variable is a program variable that is referenced in an SQL statement. Host variables are used to receive data retrieved from the database and to supply data to be added to the database.

**CALL:** The SQL statement that invokes an external routine (program) using a remote procedure paradigm. Input values are passed from CA-IDMS to the external program; the output values are returned into the host variables specified in the procedure reference.

**Bulk processing:** Bulk processing is a CA-IDMS extension to ANSI-standard SQL that allows the program to select, fetch, or insert a group of rows using a host variable array.

**Temporary table:** An application program can create a temporary table, populate it, and manipulate the data in it. A temporary table exists only for the duration of the SQL transaction in which it is created.

**Prepared statement:** A CA-IDMS extension of ANSI-standard SQL allows a program to **prepare**, or compile, certain SQL statements at runtime. This allows the program to execute an SQL statement that is not known until runtime.

## 1.2.2 Integrity constraints

Integrity rules are enforced by the DBMS using **constraints** that are specified as part of the database definition.

**Unique constraint:** A unique constraint requires that each row of a table be unique with respect to the value of a column or combination of columns. A unique constraint is defined when an index or CALC key is defined with the UNIQUE parameter.

It is possible to define any number of unique constraints on a table.

**Primary key:** The primary key is a column or combination of columns for which a unique constraint has been defined and which has been defined as not null. Consequently, the primary key uniquely identifies each row and prevents duplicate rows from being stored. For example, in the DEPARTMENT table of the demonstration database, DEPT\_ID is the primary key.

A table usually has one and only one primary key.

**Referential constraint:** A referential constraint is a relationship between two tables. A referential constraint identifies a **foreign key** in one of the tables, the **referencing** table. A foreign key is a column or combination of columns whose value must exist as the value of the primary key in a row of the related table, the **referenced** table.

When a referential constraint has been created, a row cannot be stored in the referencing table unless its foreign key value already exists as a primary key in the referenced table. Conversely, a row in the referenced table cannot be deleted or have its primary key value altered if the primary key value exists as a foreign key in the referencing table. This assures referential integrity between the tables.

**Referential constraint illustration:** The example below identifies two referential constraints between the DEPARTMENT table and the EMPLOYEE table:

1. A value cannot be stored in the DEPT\_ID column of the EMPLOYEE table unless the value exists in the DEPT\_ID column of the DEPARTMENT table
2. A value cannot be stored in the DEPT\_HEAD\_ID column of the DEPARTMENT table unless the value exists in the EMP\_ID column of the EMPLOYEE table

**DEPARTMENT table**

DEPT_ID	DEPT_NAME	DIV_CODE	DEPT_HEAD_ID
3510	APPRAISAL - USED CARS	D02	3082
4500	HUMAN RESOURCES	D09	3222
2210	SALES - NEW CARS	D04	2010
5000	CORPORATE ACCOUNTING	D09	2466
3520	APPRAISAL NEW CARS	D04	3769
4600	MAINTENANCE	D06	2096
4200	LEASING - NEW CARS	D04	1003
5100	BILLING	D06	2598
6000	LEGAL	D09	1003
1100	PURCHASING - USED CARS	D02	2246
3530	APPRAISAL - SERVICE	D06	2209
5200	CORPORATE MARKETING	D09	2894
1110	PURCHASING - NEW CARS	D04	1765
3000	CUSTOMER SERVICE	D09	4321
6200	CORPORATE ADMINISTRATION	D09	2461
2200	SALES - USED CARS	D02	2180
1120	PURCHASING - SERVICE	D06	2004
4900	MIS	D09	2466

EMPLOYEE (DEPT\_ID)  
references DEPARTMENT (DEPT\_ID)

**EMPLOYEE table**

DEPT_ID	EMP_LNAME	EMP_ID	DEPARTMENT (DEPT_HEAD_ID) references EMPLOYEE (EMP_ID)
1100	FORDMAN	5008	
1100	HALLORAN	4703	
1100	HAMEL	2246	
1110	ALEXANDER	1765	
1110	WIDMAN	2106	
1120	JOHNSON	2004	
1120	JOHNSON	3294	
1120	UMIDY	2898	
1120	WHITE	3338	
2200	ALBERTINI	2180	
.	.	.	
.	.	.	
.	.	.	

**Domain constraint:** A domain constraint restricts column values and is part of the table definition. The types of domain constraint are:

- **Data type** — Restricts column values to the data type of the column (for example, INTEGER restricts column values to the set of integers)

- **Check constraint** — Restricts column values to a range of values that satisfies a search condition
- **Null constraint** — Requires that each column of a row contain an actual value and not the absence of a value

**Constraint violation:** If the DBMS detects a constraint violation when processing an SQL statement, it returns an error.

## 1.3 Accessing non-SQL defined databases

**What you can do:** CA-IDMS provides the ability to use SQL to access a non-SQL defined database. The SQL statements used to access such a database are the same as those used to access a database that is defined with SQL DDL. Programming considerations such as session management and concurrency control are also the same.

►► Refer to *CA-IDMS SQL Reference Guide* for complete documentation of access to a non-SQL defined database using SQL.

You can use a **table procedure** or a **procedure** to process non-SQL defined data in a relational way even though the data does not conform to the rules established for such access.

A **table procedure** is a user-written program which allows any data accessible through CA-IDMS to be viewed and processed as a table. The parameters passed to and from the program are treated as the columns of a table which can be manipulated using SQL DML commands. The specifics of how the database is accessed in servicing these requests is hidden within the table procedure. A table procedure can:

- Provide full update capability on member records that do not contain foreign keys
- Access data with multiple definitions
- Access data that does not conform to the data type defined in the non-SQL schema
- Translate special data values into null values

►► Refer to *CA-IDMS SQL Reference Guide* for more information about how to use table procedures to access non-SQL databases.

A **procedure** is a user-written program and can be used to process and access a non-SQL-defined database. Procedures provide a method for implementing the remote call procedure paradigm.

When a procedure is invoked, it is called only once for each set of input values regardless of the type of statement containing the procedure reference. Within the single call, the procedure must use the input values, perform the expected action, and return the appropriate output values. This differs from a table procedure which can be called multiple times for a given set of input values depending upon the type of statement containing the procedure reference. Procedures are much easier to write and to interface with than table procedures.

►► Refer to *CA-IDMS SQL Reference Guide* for more information about using procedures to access non-SQL databases.

**Requirements** Before you can access a non-SQL defined database through SQL, you must define a schema with the SQL statement CREATE SCHEMA that references the non-SQL defined schema. Then you can reference the records defined in the non-SQL defined schema as tables in SQL DML statements.

**Tables and columns:** Once an SQL schema has been defined that references a non-SQL defined schema, each record in the non-SQL defined schema is represented as a table and each record element is represented as a column. Some elements, such as group elements, do not appear as columns in tables representing non-SQL defined records.

These transformations are applied to the names of tables and columns:

- All hyphens ('-') are translated to underscores ('\_')
- Elements occurring a fixed number of times are suffixed with an occurrence count to distinguish individual items

**Conditions imposed by database design:** The design of your non-SQL defined database may impose conditions on the use of some SQL DML statements:

- INSERT, UPDATE and DELETE statements are governed by the rules of referential integrity if the table being operated on represents a record that participates in a set defined with primary and foreign keys in the non-SQL defined schema
- When joining two tables representing records linked through a set in which the member record does not physically contain the owner's key value (that is, there are no embedded foreign keys), you must specify the set name in the join criteria

**Limitations imposed by database design:** The design of your non-SQL defined database may impose limitations on the use of some SQL DML statements:

- DELETE of a table row representing a record occurrence is disallowed if that record occurrence is the owner of any non-empty set
- INSERT is disallowed on a table representing a record if that record participates in an automatic set for which foreign keys have not been defined in the non-SQL defined schema

## 1.4 SQL application development

Given the design of the database and the application, and the description of the data, you take these steps to develop an SQL application in the CA-IDMS environment:

1. Design the application
2. Model the database access using SQL submitted through the command facility
3. Write the application
4. Create executable modules
5. Execute the application
6. Test and debug the application

### 1.4.1 Writing the application

**Program language:** In the program language, you write everything that the application program requires except database access and the structures needed to handle database access. Embedding SQL in the program does not affect any rules that apply to using the program language.

**Embedded SQL:** Within the application program, you can embed SQL statements to:

- Access the database
- Access the dictionary
- Define the structures needed to transfer data between the program and the DBMS
- Manage SQL sessions and transactions

►► Refer to the *CA-IDMS SQL Reference Guide* for complete syntax for all CA-IDMS SQL statements.

### 1.4.2 Creating executable modules

Since the application program contains an embedded sublanguage, you precompile the program to create a module of the SQL statements (an RCM) that is separate from program source. You also create an access module that contains an optimized access strategy for the SQL statements in one or more RCMs.

**Precompiling the program:** The precompiler converts embedded SQL statements to internal form and stores them in a module called an RCM. It replaces embedded SQL in the source module with calls to the DBMS. These calls, unlike the SQL statements they replace, are intelligible to the language compiler.

The precompiler checks the syntax of the embedded SQL. If there are syntax errors, it issues an error report instead of storing the RCM.

**Compiling the program:** After the program precompiles successfully, you compile and link the modified source program to create an executable program load module.

**Creating an access module:** The load module that is executed when the program requests database access is the access module. You must create the access module before executing the program.

An access module is built using one or more RCMs. Each RCM represents the SQL statements from a single source program or CA-ADS dialog.

When you create an access module, the optimizer performs these functions on each SQL statement from each RCM that you include in the access module:

- Validates table and column references in the statement against the dictionary
- Selects the most efficient database access strategy for the statement

**What information the optimizer uses:** To develop an optimized access strategy for an SQL statement, the optimizer considers:

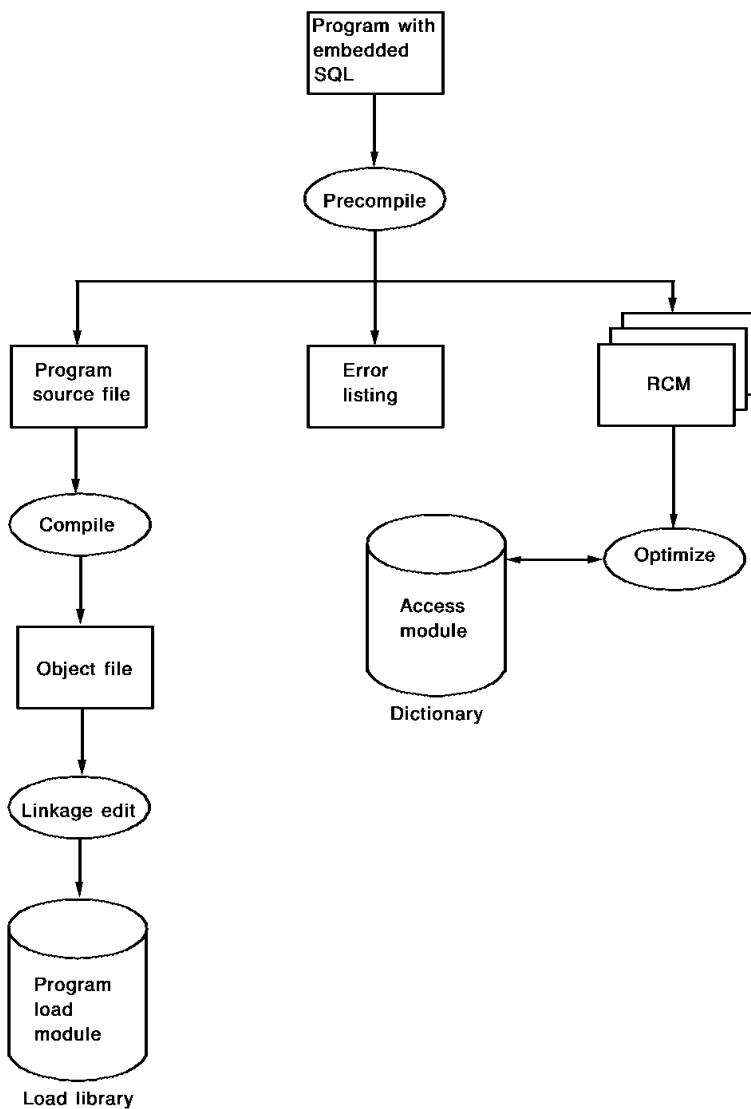
- The type of statement
- The selection criteria
- The physical structure of the database as defined in the dictionary
- Statistics stored in the dictionary as a result of running the UPDATE STATISTICS utility

**Summary of program preparation:** These are the steps you take to make the application executable:

1. Precompile the programs
2. Compile and link the programs
3. Create the access module

►► See Chapter 5, “Preparing and Executing the Program,” for detailed information about how you take these steps.

The flow chart below shows the result of each step in the process:



### 1.4.3 Executing the application

**SQL statement processing:** When the program executes at runtime, the program load module and access module are loaded as necessary. The access module is loaded the first time the program calls the DBMS to access data in the database.

The DBMS attempts to validate the definition of a table to be accessed — that is, it verifies the table definition has not changed since the access module was created. If validation fails, the DBMS automatically recreates the access module if you have defined the access module to allow this.

**Concurrency control:** When the application executes in a multiuser processing environment, the DBMS controls concurrent access to the same set of data by setting **retrieval** or **update locks**. The DBMS determines the type, level, and duration of the lock from the activities and the **isolation level** of the database transaction.

The CA-IDMS defaults for locking favor the greatest possible concurrency that can be maintained while guaranteeing the integrity of the data. You can change the system defaults for locking by specifying a different isolation level and/or a different **ready mode** for an accessed area.

- See 2.6, “Concurrency control and isolation levels” on page 2-13, for information about specifying isolation level and ready mode.

**Execution environments:** CA-IDMS application programs can execute in the DC/UCF region, a batch region, or other region such as a CICS region. Except for a local mode job, all processing of SQL statements occurs under the **central version**, the DC system component that manages multiuser, concurrent access to the database.

**Local mode** is a single-user batch processing environment that manages access to areas of the database independent of the central version. It is normally used for retrieval-only batch jobs and large-volume update applications that tend to monopolize an area of the database.

The central version performs automatic recovery for programs that end abnormally. No automatic recovery is performed for a local mode program.

#### 1.4.4 Testing and debugging the application

**Testing SQL access:** You can use the CA-IDMS Command Facility to test SQL statements online and to verify conditions of the database. When you successfully test a statement, you can save it in the dictionary.

- Refer to *CA-IDMS Command Facility* for information about using the Command Facility.

**Debugging embedded SQL:** Besides using CA-IDMS debugging tools for the host language program, you can debug embedded SQL by:

- Displaying values in fields of SQL Communication Areas (SQLCAs), where the DBMS returns information about the executing program and about SQL statement execution
  - See 2.7, “SQL status-checking and error-handling” on page 2-15, for information about displaying SQLCA fields.
- Requesting a trace of all SQL commands issued from a batch application
  - See 5.7.2, “SQL trace facility” on page 5-19, for information about the SQL trace facility.



## Chapter 2. Writing an SQL Program

---

2.1 About this chapter . . . . .	2-3
2.2 Host variables . . . . .	2-4
2.2.1 SQL declare sections . . . . .	2-5
2.2.2 INCLUDE TABLE directive . . . . .	2-5
2.2.3 Referring to host variables . . . . .	2-7
2.3 SQL sessions . . . . .	2-8
2.3.1 Beginning and ending an SQL session . . . . .	2-8
2.3.2 Session management options . . . . .	2-9
2.4 SQL transactions . . . . .	2-10
2.4.1 Beginning and ending an SQL transaction . . . . .	2-10
2.5 Effect of teleprocessing statements . . . . .	2-11
2.6 Concurrency control and isolation levels . . . . .	2-13
2.7 SQL status-checking and error-handling . . . . .	2-15
2.7.1 The SQLCA . . . . .	2-15
2.7.2 Displaying SQL Communication Area fields . . . . .	2-20
2.7.3 Error handling . . . . .	2-21
2.7.4 Checking specific errors . . . . .	2-21



## 2.1 About this chapter

This chapter discusses how a program manages SQL access to one or more databases, including these topics:

- Host variables — Variables that can be referenced in SQL statements
- SQL transaction — A database transaction initiated by an SQL statement
- SQL session — A connection to a dictionary that enables SQL access to a database
- SQL Communications Areas — Data structures the program uses to check the status of SQL statement execution uses to check the status of SQL statement execution

## 2.2 Host variables

**About host variables:** A host variable is a program variable that is referenced in an SQL statement. It is the only kind of variable that you can use in an SQL statement.

Host variables are necessary for the program to receive data from the database and in most cases for the program to modify data in the database.

**How host variables are used:** Host variables are used to:

- Receive column values specified in a SELECT or FETCH statement
- Supply column values specified in an UPDATE statement, INSERT statement, or other statements containing a search condition

**Host variable example:** In this example, DEPT-ID, EMP-LNAME, and EMP-ID are host variables. DEPT-ID and EMP-LNAME receive column values and EMP-ID supplies a column value used in the search condition of the statement:

```
EXEC SQL
  SELECT DEPT_ID,
    EMP_LNAME
  INTO :DEPT-ID,
    :EMP-LNAME
  FROM EMPLOYEE
  WHERE EMP_ID = :EMP-ID
END-EXEC.
```

**Indicator variable:** An indicator variable is a host variable used to manipulate null values.

CA-IDMS sets an indicator variable to -1 if the column value in the associated host variable is null.

An indicator variable should be defined for each column accessed by the program that could contain a null value. If the program retrieves a null value from a column that has no indicator variable, CA-IDMS returns an error.

In a host variable array for use in bulk processing, the data type of an indicator variable must be declared with a usage SQLIND.

**Null value:** A null value is the absence of a value and is not the same as spaces or numeric zeros, which are actual values. In an SQL-defined database, a column, regardless of data type, can contain a null value unless the column definition specifically disallows them.

## 2.2.1 SQL declare sections

**About declare sections:** In ANSI-standard SQL, you define host variables within an SQL declare section. You begin and end an SQL declare section with these statements:

```
EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC.
.
.
EXEC SQL
  END DECLARE SECTION
END-EXEC.
```

A CA-IDMS extension of ANSI-standard SQL allows you to continue an SQL declaration section statement on the following line after any keyword.

**What you can do:** You can include any number of host variable declarations in an SQL declare section. You can include any number of SQL declare sections in a single application program.

**Host variable declaration example:** In this example, the SQL declare section defines host variables, including one indicator variable, using standard COBOL data declarations.

```
WORKING-STORAGE SECTION.
.
.
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 EMP-ID          PIC S9(8)      USAGE COMP.
  01 EMP-LNAME       PIC X(20).
  01 SALARY-AMOUNT   PIC S9(6)V(2)  USAGE COMP-3.
  01 PROMO-DATE      PIC X(10).
  01 PROMO-DATE-I    PIC S9(4)      USAGE COMP.
EXEC SQL END DECLARE SECTION END-EXEC.
```

## 2.2.2 INCLUDE TABLE directive

**The INCLUDE TABLE statement:** You can use the INCLUDE TABLE statement, a CA-IDMS extension of ANSI-standard SQL, to define a host language data structure for table columns. INCLUDE TABLE is a precompiler directive that defines host variables for all columns of a table or view, or for a subset of columns that you specify in the statement.

If INCLUDE TABLE falls within the scope of an SQL declare section, embedded SQL statements can reference the variables defined by the precompiler as host variables.

**Statement example:** The INCLUDE statement below directs the precompiler to define host variables for the DIVISION table, which has columns DIV\_CODE, DIV\_NAME, and DIV\_HEAD\_ID:

```
WORKING-STORAGE SECTION.  
.  
.EXEC SQL  
    INCLUDE TABLE DIVISION  
END-EXEC.
```

**Structure example:** When the precompiler processes the INCLUDE TABLE statement in the above example, it defines this structure:

```
*EXEC SQL  
*   INCLUDE TABLE DIVISION  
*END-EXEC.  
01  DIVISION.  
    03  DIV-CODE          PIC X(3).  
    03  DIV-HEAD-ID       PIC S9(4) COMP.  
        03  DIV-HEAD-ID-I  COMP PIC S9(8).  
    *  
    03  DIV-NAME.  
        49  DIV-NAME-LEN   PIC S9(4) COMP.  
        49  DIV-NAME-TEXT  PIC X(40).
```

**INCLUDE statement options:** You can use options on the INCLUDE statement to:

- Override the default element level
- Direct the precompiler not to group elements under a structure
- Specify the columns to be included
- Specify names for the generated record and element definitions
- Specify a prefix and/or suffix for an element name
- Direct the precompiler to generate a multiply-occurring array

►► Refer to the *CA-IDMS SQL Reference Guide* for documentation of INCLUDE statement syntax and options.

**Including an array:** You can use the include statement to generate a host variable array by specifying the NUMBER OF ROWS parameter. A host variable array is used in bulk processing.

►► See 3.4, “Bulk processing” on page 3-19, for more information about bulk processing.

**Host variable array structure:** When the precompiler generates a host variable array, it creates a structure using three levels. In the example below, a structure has been generated by an INCLUDE TABLE statement with NUMBER OF ROWS = 100:

```

01 DIVISION.
02 DIVISION-BULK OCCURS 100 TIMES.
  03 DIV-CODE          PIC X(3).
  03 DIV-HEAD-ID       PIC S9(4) COMP.
  03 DIV-HEAD-ID-I    COMP PIC S9(8).
*
  03 DIV-NAME.
    49 DIV-NAME-LEN    PIC S9(4) COMP.
    49 DIV-NAME-TEXT   PIC X(40).

```

**Usefulness of INCLUDE TABLE:** The INCLUDE TABLE statement is a programming tool. It assures that host variable definitions correspond to current table column definitions in the dictionary: the data types are equivalent, and indicator variables are declared for all columns that allow null values.

**When not to use INCLUDE TABLE:** Using INCLUDE TABLE is not appropriate if:

- The program must conform to ANSI-standard SQL
- The host variable declaration is for temporary table columns

### 2.2.3 Referring to host variables

**Reference requirements:** These syntax requirements apply when you refer to a host variable in an embedded SQL statement:

- To refer to any host variable in an embedded SQL statement, prefix the host variable name with a colon (:)
- To associate an indicator variable with a host variable, place the reference to the indicator variable after the host variable, with *no comma* or other separator character

**Note:** You can use the optional keyword INDICATOR as a separator.

**Reference example:** In the example below, information from the BENEFITS table is selected for a given employee ID value, which the program has assigned to the host variable EMP-ID. BENEFITS table information is retrieved into host variables VAC-TAKEN and SICK-TAKEN. VAC-TAKEN-I and SICK-TAKEN-I are indicator variables.

```

EXEC SQL
  SELECT VAC_TAKEN,
         SICK_TAKEN
    INTO :VAC-TAKEN INDICATOR :VAC-TAKEN-I,
         :SICK-TAKEN INDICATOR :SICK-TAKEN-I
   FROM BENEFITS
 WHERE EMP_ID = :EMP-ID
END-EXEC.

```

## 2.3 SQL sessions

An SQL session is a logical connection between the executing application and the DBMS. It begins when the application connects to a dictionary and ends when the application disconnects from the dictionary. The dictionary contains the definition of the data accessed using SQL.

### 2.3.1 Beginning and ending an SQL session

**Automatic connection:** An SQL session begins automatically by establishing a dictionary connection when the program submits the first SQL statement. The program connects to the default dictionary in effect for the user session.

**Default dictionary:** When establishing an automatic connection, CA-IDMS uses the default dictionary value associated with the user session.

CA-IDMS determines the default dictionary by:

- The SYSIDMS DICTNAME parameter, for local mode access
  - See the *CA-IDMS Database Administration* guide for information about SYSIDMS parameters.
- The value for the DICTNAME attribute for the user session, as set by one of the following:
  - The user profile
  - The system profile
  - A DCUF SET DICTNAME statement

**Explicit connection:** The program can start an SQL session by establishing a connection to a specified database with a CONNECT TO statement.

In this example, the CONNECT TO statement specifies a host variable containing the name of the dictionary for the SQL session:

```
EXEC SQL
    CONNECT TO :DICT-NAME
END-EXEC.
```

**When to use CONNECT TO:** The program should issue a CONNECT TO statement to access a database that is defined in a dictionary other than the default dictionary for the user session.

**SQL statements that end a session:** If the SQL session began automatically (that is, no CONNECT TO statement was issued), it ends when the program issues one of these statements:

- COMMIT

- ROLLBACK
- COMMIT RELEASE
- ROLLBACK RELEASE
- RELEASE

If a CONNECT statement was executed, the program must explicitly end the SQL session with one of these statements:

- COMMIT RELEASE
- ROLLBACK RELEASE
- RELEASE

**Automatic session termination:** If the application program terminates execution by returning control to the operating system or TP monitor, all SQL sessions still in progress are terminated automatically as if the application had issued a ROLLBACK RELEASE statement.

### 2.3.2 Session management options

**Suspending a session:** A program can suspend the current SQL session by issuing a SUSPEND SESSION statement. The program can reactivate the suspended session by issuing a RESUME SESSION statement.

Using SUSPEND SESSION and RESUME SESSION allows a program to release resources but retain the ability to reestablish the session in the same state as when it was suspended.

These session management statements are a CA-IDMS extension of ANSI-standard SQL specifically designed for pseudoconversational programming.

►► See 6.3, “Pseudoconversational programming” on page 6-9, for information about how to develop a pseudoconversational program with embedded SQL.

**Concurrent sessions:** A program can maintain multiple concurrent sessions.

►► The technique for managing multiple concurrent sessions is discussed in 6.4, “Managing concurrent sessions” on page 6-15.

## 2.4 SQL transactions

An SQL transaction is a database transaction — a unit of recovery — within an SQL session.

### 2.4.1 Beginning and ending an SQL transaction

**Beginning a transaction:** CA-IDMS begins an SQL transaction when the program submits an SQL statement that results in access to either user data or the dictionary.

►► Refer to the *CA-IDMS SQL Reference Guide* for a list of SQL statements that start an SQL transaction.

**Ending a transaction:** An SQL transaction ends when:

- A COMMIT statement is executed
- A ROLLBACK statement is executed
- The SQL session is terminated

**Note:** If the SQL session is terminated while the transaction is in progress, all uncommitted updates made by the transaction are rolled back.

When a transaction is ended, all open cursors are closed, all temporary tables are dropped, and all prepared statements are dropped.

►► See 3.3, “Using a cursor” on page 3-12, for information about cursors. See 6.5, “Creating and using a temporary table” on page 6-18, for information about temporary tables. See 7.4, “Executing prepared SELECT statements” on page 7-10, for information about prepared statements.

**Continuing the transaction:** A CA-IDMS extension of ANSI-standard SQL allows you to commit updates without ending the transaction. This extension is the CONTINUE parameter of the COMMIT statement:

```
EXEC SQL  
    COMMIT CONTINUE  
END-EXEC.
```

The CONTINUE parameter limits the effect of COMMIT to committing updates and downgrading or releasing update locks held for the transaction.

**Committing changes:** In an SQL transaction changed data is *not* automatically committed, even when the transaction ends normally. The program must issue a form of the COMMIT statement to commit changed data.

## 2.5 Effect of teleprocessing statements

**Effect of DC statements:** Statements used to manage application processing in the CA-IDMS environment can affect the status of an SQL transaction or session, as The following table shows:

Statement	Effect
COMMIT TASK	COMMIT CONTINUE on all nonsuspended SQL sessions and database transactions
ROLLBACK TASK	ROLLBACK RELEASE on all nonsuspended SQL sessions and database transactions
COMMIT TASK ALL	COMMIT on all nonsuspended SQL sessions and database transactions
FINISH TASK	COMMIT RELEASE on all nonsuspended SQL sessions and database transactions
ROLLBACK TASK CONTINUE	ROLLBACK on all nonsuspended SQL sessions and database transactions
Normal task termination	ROLLBACK RELEASE on all nonsuspended SQL sessions
Abnormal task termination	ROLLBACK RELEASE on all SQL sessions

**Effect of CA-ADS application termination:** The termination of a CA-ADS application causes a ROLLBACK RELEASE to be issued on all SQL sessions.

**Effect of CICS statements:** The effect of a CICS statement on an SQL session depends on the parameters used to generate the version of IDMSCINT that was link edited with the issuing application:

Statement	Effect
SYNCPOINT	<ul style="list-style-type: none"> <li>■ If IDMSCINT was generated with <b>AUTOCOMT=NO</b> (the default), a SYNCPOINT request has no impact on SQL sessions and transactions</li> <li>■ If IDMSCINT was generated with <b>AUTOCOMT=YES</b>: <ul style="list-style-type: none"> <li>– A SYNCPOINT COMMIT request is translated into a COMMIT RELEASE on all nonsuspended SQL sessions and database transactions</li> <li>– A SYNCPOINT ROLLBACK request is translated into a ROLLBACK RELEASE on all SQL sessions and database transactions</li> </ul> </li> </ul>
Normal CICS transaction termination	<ul style="list-style-type: none"> <li>■ If IDMSCINT was generated with <b>AUTOCOMT=NO</b> (the default), ROLLBACK RELEASE is issued on all nonsuspended SQL sessions and database transactions</li> <li>■ If IDMSCINT was generated with <b>AUTOCOMT=YES</b>, a COMMIT RELEASE is issued on all nonsuspended SQL sessions and database transactions</li> </ul>
Abnormal task termination	ROLLBACK RELEASE on all SQL sessions and database transactions

## 2.6 Concurrency control and isolation levels

**Concurrency control:** CA-IDMS manages concurrent access to the same set of data with a system of locks. The degree of concurrent access allowed by a database transaction is determined by the isolation level of the transaction and the ready mode of the areas it accesses.

**Locks:** CA-IDMS provides two types of lock:

- A **retrieval lock** prevents updates but allows retrieval of data by another database transaction
- An **update lock** prevents both updates and retrieval of data by another database transaction

**Isolation levels and locking:** CA-IDMS supports two isolation levels. The descriptions below of how the system performs locking under each isolation level assume the least restrictive ready mode for areas accessed by the database transaction:

- Cursor stability — Under cursor stability, the DBMS places a retrieval lock on the row on which an updateable cursor is positioned until the cursor position changes. It places a retrieval lock on the row accessed by a SELECT statement that accesses only one row (a single-row select) until the SQL transaction accesses another row from the same table. It releases update locks when the transaction either terminates or issues a COMMIT CONTINUE.
- Transient read — Under transient read, the DBMS:
  - Places no locks on rows accessed by the transaction
  - Allows the transaction to retrieve locked rows
  - Prevents the transaction from performing updates

**Concurrency under cursor stability:** Cursor stability provides the greatest possible concurrency while guaranteeing the integrity of data read by the transaction. Under cursor stability:

- The row on which an updateable cursor is positioned cannot be updated by another database transaction before the cursor position changes
- A single row retrieved by a SELECT statement cannot be updated by another database transaction until the original transaction accesses another row of the table

Cursor stability is the CA-IDMS default. It is appropriate for high-volume transaction environments.

**Concurrency under transient read:** Transient read provides no protection from the effects of concurrent database transactions. It allows a database transaction to read data that has not been committed and allows concurrent database transactions to update the data.

Transient read is appropriate when the transaction is retrieval only and does not require the data to be consistent and entirely accurate.

**Specifying the isolation level:** You can specify the default isolation level with the DEFAULT ISOLATION parameter of the CREATE ACCESS MODULE statement.

- See 5.4, “Creating the access module” on page 5-11, for further information about how to create the access module.

A program can override the default isolation level for the access module by issuing a SET TRANSACTION statement. The specification on this statement remains in effect until the end of the transaction.

- Refer to *CA-IDMS SQL Reference Guide* for documentation of the SET TRANSACTION statement.

**Area ready mode:** You can control concurrent access at the area level using the READY parameter of the CREATE ACCESS MODULE statement. This parameter allows you to specify what type of retrieval or update lock the DBMS sets on an area that the program accesses. The type of lock, in combination with the PRECLAIM or INCREMENTAL option, determines how long the DBMS holds the lock for the transaction.

- Refer to *CA-IDMS SQL Reference Guide* for documentation of the READY parameter of the CREATE ACCESS MODULE statement.

**Repeatability:** If you specify a ready mode of protected retrieval or protected update, the DBMS will prevent concurrent update access in the specified areas for the duration of a database transaction. This gives the transaction running under cursor stability the ability to repeat a read of the specified area or areas without changes to the data by other transactions.

- Refer to *CA-IDMS Database Administration* for documentation of the lock management system.

## 2.7 SQL status-checking and error-handling

When CA-IDMS executes an SQL statement, it returns information about the status of statement execution to a data structure called the SQLCA. Your program should contain logic to handle exceptional conditions resulting from statement execution. This logic takes the form of checking SQLCA information.

### 2.7.1 The SQLCA

**About the SQLCA:** The SQL Communication Area (SQLCA) is a data structure to which the DBMS returns information about the execution of an SQL statement.

**SQLSTATE:** SQLSTATE is a five-character string in which CA-IDMS returns the status of the last SQL statement executed. It is divided into a two-character class and a three-character subclass. Standard values are associated with each class and subclass, which minimizes the need for vendors to define their own values and makes applications more portable from one environment to another.

The list of SQLSTATE values that CA-IDMS can return appear below. The list is divided into sections based on the class (the first 2 characters of the SQLSTATE value). Each subclass (the last 3 characters of the SQLSTATE value) is listed under its associated class.

- **ANSI- and ISO-defined values** — Class and subclass values beginning with the characters A-H and 0-4 are established by the ANSI and ISO standards organizations.
- **CA-IDMS-defined values** — Class and subclass values beginning with the characters I-Z and 5-9 are vendor-defined; in this case, they are specific to CA-IDMS. (Any subclass value associated with a vendor-defined class is also defined by that vendor.)

```
00 Successful completion
 000 No subclass

01 Warning
 000 No subclass
 004 String data, right truncation
 600 Inconsistent or invalid option
 602 Entity or association already exists
 605 Entity not defined in Catalog
 606 Invalid option for physical DDL
 607 Invalid option for DMCL
 608 Connecting to a dictionary which is missing either or
      or both of DDLCAT/DDLDML areas
 610 Database is inconsistent with request
 638 Warning returned from table procedure

02 No data
 000 No subclass

07 Dynamic SQL error
 000 No subclass
 001 USING clause does not match dynamic parameter specification
 002 USING clause does not match target specification
 003 Cursor specification cannot be executed
 004 USING clause required for dynamic parameters

08 Connection exception
 000 No subclass
 004 SQL-server rejected establishment of SQL-connection
 006 Connection failure

21 Cardinality violation
 000 No subclass

22 Data Exception
 000 No subclass
 001 String data, right truncation
 002 Null value, no indicator parameter
 003 Numeric value out of range
 005 Error in assignment
 007 Invalid datetime format
 008 Datetime field overflow
 011 Substring error
 012 Division by zero
 019 Invalid escape character
```

```
23 Constraint violation
  000 No subclass
  501 Duplicate key violation

24 Invalid cursor state
  000 No subclass

25 Invalid transaction state
  000 No subclass
  006 Read-only SQL-transaction

26 Invalid SQL statement name
  000 No subclass

28 Invalid authorization specification
  000 No subclass
  602 Entity or association already defined
  605 Entity or association not previously defined
  607 Authorization ids not specified

2C Invalid character set name
  000 No subclass

38 External routine exception
  000 No subclass

39 External routine invocation exception
  000 No subclass

3F Invalid schema name
  000 No subclass

40 Transaction rollback
  000 No subclass
  001 Serialization failure

42 Syntax error or access rule violation
  000 No subclass
  500 Table not found
  501 Column not found
  502 Entity already defined
  503 Authorization failure
  504 Cursor not declared or previously declared
  505 Entity not found
  506 Invalid identifier
  507 Keyword used as identifier
  600 Invalid statement
  601 Statement not valid in this context
  603 Statement not valid for this schema
  604 Invalid data type
  606 Invalid statement option
  607 Missing statement option
  609 Invalid constraint definition
  610 Invalid number of columns
```

```

50 CA-defined errors
 000 No subclass
 002 Limit exceeded
 003 Space exceeded
 00B Internal error
 00I Schema mismatch
 00J Invalid entity definition
 00K Uncategorized error
 00L Invalid calling parameters

60 CA-IDMS specific errors
 000 No subclass
 001 Problem with load module or synchronization stamps
 002 Database error
 003 Rollback failed

64 CA-IDMS Physical DDL error
 000 No subclass

6U CA-IDMS Utility error
 000 No subclass

```

**SQLCODE:** For status-checking, another important field in the SQLCA structure is SQLCODE. The table below shows the values that the DBMS may return to this field.

Value	Meaning
< 0	The SQL statement returned an error (see error values below)
0	The SQL statement was executed successfully
1	The SQL statement was executed successfully with a warning
100	There are no more rows associated with the current query, or no rows satisfied the search criteria in a searched update or delete

**Note:** The ANSI and ISO standards only define meaning to the values of **0** and **100**. Negative SQLCODE values signify an error; however, specific values are not standardized as with SQLSTATE.

**SQLCODE error values:** The table below associates SQLCODE error values with one of the three kinds of SQL statement failure and suggests the appropriate error-handling strategy for each category of error:

<b>Value</b>	<b>Level of failure</b>	<b>Meaning</b>
-7	Task	An internal error caused a task abend, leading to rollback and termination of the SQL transaction and termination of the SQL session
-6	SQL session	An error caused an SQL session failure, leading to rollback, termination of the SQL transaction and termination of the SQL session.  A program intending to retry the SQL statements should first terminate the SQL session with one of these statements: <ul style="list-style-type: none"> <li>■ ROLLBACK RELEASE</li> <li>■ RELEASE</li> <li>■ The equivalent TP monitor command</li> </ul>
-5	SQL transaction	An error has caused an SQL transaction failure, leading to rollback and termination of the SQL transaction.  A program intending to retry the SQL statements should first terminate the transaction with one of these statements: <ul style="list-style-type: none"> <li>■ ROLLBACK</li> <li>■ ROLLBACK RELEASE</li> <li>■ RELEASE</li> <li>■ The equivalent TP monitor command</li> </ul>
-4	SQL statement	An error has caused failure of the SQL statement to execute; the effect of the statement, if any, on the database has been rolled out.  Unless the reason for the error is one that the program can handle, the program should terminate the session or transaction.

**SQLCERC:** If an error is returned, the DBMS also returns a value in the SQLCERC field of the SQLCA. The value in this field is the SQL error code.

In certain cases, you can use this information to recover from error conditions. For example, if 1038 is returned to SQLCERC, a deadlock has occurred. The application program can handle the deadlock by first terminating the database transaction and then resuming processing after the last commit or start of transaction.

SQLCERC values correspond to the last four digits of the CA-IDMS DB runtime messages. To determine the meaning of a particular SQLCERC value, refer to the text and description of the equivalent DB message.

- Refer to *CA-IDMS Messages and Codes*, or issue a DCMT DISPLAY MESSAGE DBnnnnnnn statement, as documented in *CA-IDMS System Tasks and Operator Commands*, for documentation of DB messages.

**Other SQLCA fields:** For error-checking and reporting, these are other useful SQLCA fields:

Field	Description of contents
SQLCLNO	Source file line number from which the SQL statement was obtained
SQLCSER	Offset into the SQL statement buffer where a syntax error was recognized
SQLCNRP	Number of rows processed by the SQL statement
SQLCERM	Text of the error message
SQLCERL	Length of error message text

**How SQLCA is initialized:** The DBMS initializes SQLCA values on every SQL statement execution. If the program accesses the SQLCA after issuing an SQL statement, all SQLCA values refer to that statement.

**SQLPIB fields:** When you display or log error information, you may wish to include information in fields of the SQL Program Information Block (SQLPIB):

Field	Description of contents
SQLDTS	Date and time the program was compiled
SQLPGM	Name of the RCM that is the source of the SQL statement

- Refer to the *CA-IDMS SQL Reference Guide* for descriptions of SQLPIB fields.

## 2.7.2 Displaying SQL Communication Area fields

**SQLCA structure:** The technique used by the program to access and display SQLCA information may depend on the SQLCA structure and the rules governing use of the host program language.

For example, in the COBOL SQLCA structure, SQLCODE is defined as PIC S9(9) USAGE COMPUTATIONAL. To display any possible SQLCODE value, including a

negative value, you should first move the SQLCODE value to a work field defined as PIC -9(4).

► See Chapter 4, “Requirements and Options for Host Languages,” for information about the SQLCA structure.

**Displaying an SQL message:** To display an SQL error message, you use the IDMSIN01 entry point to the IDMS module to call a function that formats error message data using information in the SQLERM and SQLCERL fields.

► See Appendix D, “Calls to IDMSIN01,” for information about the requirements for calling IDMSIN01 to display SQL messages.

### 2.7.3 Error handling

**Using WHENEVER SQLERROR:** If the program handles most or all errors by branching to one routine, consider using the WHENEVER precompiler directive statement that specifies the SQLERROR condition. The precompiler adds the logic requested by a WHENEVER statement immediately after every SQL statement that follows the WHENEVER statement.

In this example, if an SQL statement that follows the WHENEVER statement returns an error, processing branches to the routine labeled ERROR-EXIT:

```
EXEC SQL
    WHENEVER SQLERROR GOTO :ERROR-EXIT
END-EXEC.
```

**Overriding WHENEVER:** Once the the program issues a WHENEVER SQLERROR statement, it can override the statement only with:

- Another WHENEVER SQLERROR statement that specifies different branching logic
- A WHENEVER SQLERROR CONTINUE statement, which directs the precompiler not to add logic after subsequent SQL statements

► Refer to *CA-IDMS SQL Reference Guide* for documentation of the WHENEVER statement.

### 2.7.4 Checking specific errors

**When to do it:** In certain situations the program should check for specific errors before directing processing to a generalized error routine.

This manual discusses when to code specific error-checking logic in Chapter 3, “Data Manipulation with SQL,” and other chapters that present SQL programming techniques.

**How to do it:** You write a conditional program statement to check for a specific SQL error following the SQL statement. The conditional statement must also account for all other errors if the test for the specific error fails:

```
EXEC SQL
  INSERT
    INTO DIVISION
      VALUES (:DIVISION-CODE,
              :DIVISION-NAME,
              :DIV-HEAD-ID)
END-EXEC.

IF SQLSTATE = '23501' PERFORM EXISTING-DIVISION
ELSE IF SQLCODE < 0 GOTO ERROR-ROUTINE.
```

**Using WHENEVER SQLERROR:** To perform specific error-checking after the program has issued a WHENEVER SQLERROR statement, you can:

- Override the previous WHENEVER statement before issuing the SQL statement:

```
EXEC SQL
  WHENEVER SQLERROR CONTINUE
END-EXEC.

EXEC SQL
  INSERT
  .
  .
  .
END-EXEC.

IF SQLSTATE = '23501' PERFORM EXISTING-DIVISION
ELSE IF SQLCODE < 0 GOTO ERROR-ROUTINE.

EXEC SQL
  WHENEVER SQLERROR GOTO ERROR-ROUTINE
END-EXEC.
```

- Place the specific error-handling logic in the generalized routine:

```
ERROR-ROUTINE.

IF SQLSTATE = '23501' PERFORM EXISTING-DIVISION.
.
.
.
```

# Chapter 3. Data Manipulation with SQL

---

3.1 About this chapter . . . . .	3-3
3.2 Data manipulation operations . . . . .	3-4
3.2.1 Retrieving data . . . . .	3-4
3.2.2 Adding data . . . . .	3-6
3.2.3 Modifying data . . . . .	3-7
3.2.4 Deleting data . . . . .	3-9
3.2.5 Using indicator variables in data manipulation . . . . .	3-10
3.3 Using a cursor . . . . .	3-12
3.3.1 Declaring a cursor . . . . .	3-12
3.3.2 Fetching a row . . . . .	3-13
3.3.3 Executing a positioned update or delete . . . . .	3-15
3.4 Bulk processing . . . . .	3-19
3.4.1 Executing a bulk fetch . . . . .	3-19
3.4.2 Executing a bulk select . . . . .	3-22
3.4.3 Executing a bulk insert . . . . .	3-23
3.5 Invoking external routines . . . . .	3-25
3.5.1 Call Statement . . . . .	3-25
3.5.1.1 CALL of a procedure . . . . .	3-25
3.5.1.2 CALL of a table procedure . . . . .	3-25
3.5.2 SELECT statement . . . . .	3-26
3.5.2.1 SELECT of a procedure . . . . .	3-26
3.5.2.2 SELECT of a table procedure . . . . .	3-26



## 3.1 About this chapter

This chapter shows you how to perform data manipulation with SQL in a host language program and suggests ways that the program can take advantage of SQL DML in CA-IDMS. This includes discussion of:

- Retrieving data
- Storing and modifying data
- Deleting data
- Managing multiple rows
  - Using a cursor
  - Bulk processing
- Appropriate status-checking
- Invoking external routines

## 3.2 Data manipulation operations

**SQL DML statements:** You use these SQL statements in data manipulation operations:

- SELECT — To retrieve data
- INSERT — To add data
- UPDATE — To modify data
- DELETE — To delete data

SQL data manipulation statements provide these capabilities:

- One statement can manipulate data in many rows
- One statement can perform both computation and data manipulation
- One statement can retrieve data from many tables

This means that you have several options for performing each type of data manipulation.

### 3.2.1 Retrieving data

**Using SELECT:** In a program, you use the SELECT statement in one of these ways to retrieve data from the database:

- With an INTO clause that specifies **host variable** names, to retrieve a single row into working storage
- With a BULK clause that specifies the name of a **host variable array**, to retrieve multiple rows into working storage
- In a DECLARE CURSOR statement to define a **cursor** that you can use to retrieve multiple rows and then fetch each row one at a time into working storage
- In an INSERT statement to select from one or more other tables the data to be inserted

When embedding a SELECT statement, specify each column even if you mean to select all columns. Using SELECT \* to select all columns can cause a program error if, for example, a column is added to the table.

**Single-row SELECT statement:** If the result of a SELECT statement will be one and only one row, you can issue a SELECT statement with an INTO clause.

A result table will contain only one row when:

- The WHERE clause specifies a primary key value as the search condition:

```
EXEC SQL
  SELECT EMP_ID,
         EMP_LNAME,
         DEPT_ID
    INTO :EMP-ID,
         :EMP-LNAME,
         :DEPT-ID
   FROM EMPLOYEE
  WHERE EMP_ID = :EMP-ID
END-EXEC.
```

- All column values result from aggregate functions and no GROUP BY clause has been specified:

```
EXEC SQL
  SELECT COUNT(P.EMP_ID) INTO :TOT-EMPLOYEES,
         SUM(B.SALARY_AMOUNT) INTO :TOT-SALARIES,
         (SUM(B.VAC_ACCRUED) - SUM(B.VAC_TAKEN))
           INTO :UNUSED-VAC
    FROM POSITION P, BENEFITS B
   WHERE P.EMP_ID = B.EMP_ID
     AND P.SALARY_AMOUNT IS NOT NULL
     AND P.FINISH_DATE IS NULL
END-EXEC.
```

**Checking single-row select status:** If the number of rows returned by a SELECT statement with an INTO clause is greater than one, the DBMS returns a *cardinality violation* error. No data is moved to the host variables named in the INTO clause.

If no row is found that matches the selection criteria, the DBMS returns a *no rows found* warning and moves 100 to SQLCODE.

**Updating the single row:** Under cursor stability if the program performs single-row select that specifies the primary key in the search condition, the DBMS locks the base row from which the resulting row is derived. This prevents any update by a concurrent database transaction. The lock is maintained until one of these events occurs:

- The database transaction ends
- The database session is suspended
- The database transaction accesses a different row from the same table

Until one of these events occurs, the SQL transaction can update the row without a need to check whether a concurrent transaction has modified the row.

►► See 3.2.3, “Modifying data” on page 3-7, for further information about updating rows.

**Multiple-row SELECT:** If the result table of a SELECT statement potentially has multiple rows, the program must declare a cursor or perform bulk processing to process retrieved data.

►► See 3.3, “Using a cursor” on page 3-12 or 3.4, “Bulk processing” on page 3-19, for more information on retrieving multiple rows.

### 3.2.2 Adding data

**Using INSERT:** In a program, you use an INSERT statement to add data to the database in one of these ways:

- INSERT with a VALUES clause to add a single row to a table by listing the column values in the statement
- INSERT with a SELECT statement to add one or more rows using existing data
- INSERT with a BULK clause to add multiple rows to a table from a host variable array

**Single-row INSERT:** To add a single row to a table, issue an INSERT statement with a VALUES clause that specifies a value for each column in the column list:

```
EXEC SQL
  INSERT INTO DIVISION
    (DIV_CODE, DIV_NAME)           ← Column list
    VALUES (:DIV-CODE, :DIV-NAME)
END-EXEC.
```

**Multiple-row INSERT with SELECT:** One way to add multiple rows to a table is to insert the result table of a SELECT statement.

In this example, a result table of data from the EMPLOYEE table is inserted into a table named TEMP\_MGR:

```
EXEC SQL
  INSERT INTO TEMP_MGR
    SELECT DISTINCT E.MANAGER_ID,
      M.EMP_FNAME,
      M.EMP_LNAME
    FROM EMPLOYEE E, EMPLOYEE M
    WHERE E.MANAGER_ID = M.EMP_ID
END-EXEC.
```

**Guidelines for INSERT:** Apply these guidelines when formulating an INSERT statement:

- An INSERT statement must supply a value for each column in the column list, even if the value is null
- The order of column values must match the order of the column list
- An INSERT statement must supply values for *all* columns of the named table if the column list is omitted:

```
EXEC SQL
  INSERT INTO DIVISION
    VALUES ('D06', 'ADVANCED RESEARCH', NULL)
    -- Division head id is null --
END-EXEC.
```

When embedding an INSERT statement with a VALUES clause, you should include a column list even if you mean to insert values into all columns. Using a VALUES clause but omitting a column list can cause a program error if, for example, a column has been added to the table.

- A column list must include any table columns that are defined as not null and as *not* having a default value

If an INSERT statement omits a table column from the column list, the DBMS:

- Stores the default value for the column, if one has been defined
- Stores a null value if the column allows nulls
- Returns a *data exception* error if no default value has been defined and nulls are not allowed

**Checking INSERT status:** Since the DBMS enforces integrity constraints, the program can test SQLCERC for a *constraint violation*:

- 1023 — Check constraint
- 1058 — Unique constraint
- 1060 — Referential constraint
- 1002 — Null constraint
- 1031 — Page group violation

**Note:** Indexes and referential constraints (linked and unlinked) may not cross page group boundaries.

Here is an example for a specific test for a check constraint violation:

```
IF SQLCERC = 1023 PERFORM INVALID-DATA  
ELSE IF SQLCODE < 0 GOTO ERROR-ROUTINE.
```

If an INSERT statement that uses a SELECT statement executes successfully but adds no rows, the DBMS returns 100 to SQLCODE and 0 to SQLCNRP.

**Inserting multiple rows:** You can add a set of rows to a table using one INSERT statement with a BULK clause.

►► See 3.4, “Bulk processing” on page 3-19, for information about using bulk processing to insert.

### 3.2.3 Modifying data

**Using UPDATE:** You modify data in a table using an UPDATE statement. There are two types of UPDATE statement:

- If the WHERE clause contains a search condition, the statement modifies any row that meets the search condition — this is a **searched update**
- If the UPDATE statement specifies WHERE CURRENT OF *cursor-name*, the statement modifies only the row on which the cursor is positioned — this is a **positioned update**

►► See 3.3, “Using a cursor” on page 3-12, for information about positioned updates.

**Checking UPDATE status:** As with an INSERT statement, the DBMS enforces integrity constraints when the program issues an UPDATE statement.

►► See 3.2.2, “Adding data” on page 3-6, for information about checking statement execution for constraint violation.

**Searched updates:** A searched update statement contains:

- A SET clause that specifies a value for each column to be updated
- A WHERE clause containing the criteria for choosing the rows to be updated

**Searched updates using host variables:** In this example, the UPDATE statement uses a host variable (SALARY-AMOUNT) to transfer a new data value to the database and another host variable (EMP-ID) supplies the column value that is the criterion for choosing the row to update:

```
EXEC SQL
  UPDATE POSITION
    SET SALARY_AMOUNT = :SALARY-AMOUNT
    WHERE EMP_ID = :EMP-ID
END-EXEC.
```

The statement in the example above updates only one row because the search condition is restricted by the value of a primary key (EMP\_ID).

The statement in the example below updates multiple rows if more than one employee does the job represented by the value in JOB-ID:

```
EXEC SQL
  UPDATE POSITION
    SET SALARY_AMOUNT = :SALARY-AMOUNT
    WHERE JOB_ID = :JOB-ID
END-EXEC.
```

**Searched updates without host variables:** A searched update may operate on existing column values without using host variables. This statement gives a 10 percent raise to all employees with a current salary in a specified range:

```
EXEC SQL
  UPDATE POSITION
    SET SALARY_AMOUNT = 1.1 * (SALARY_AMOUNT)
    WHERE SALARY_AMOUNT BETWEEN 20000 AND 40000
END-EXEC.
```

**No matching rows:** If no rows satisfy the selection criteria in the WHERE clause of a searched update, SQLCODE will be set to 100.

**Automatic rollback:** If the attempt to update one row of a searched update fails:

- Statement execution halts
- The DBMS returns an error value to SQLCODE
- The results of the UPDATE statement are automatically rolled back

### 3.2.4 Deleting data

**Using DELETE:** You erase rows from a table using a DELETE statement. As with UPDATE, there are two types of DELETE statement:

- If the WHERE clause contains a search condition, the statement deletes any row that meets the search condition — this is a **searched delete**
- If the DELETE statement specifies WHERE CURRENT OF *cursor-name*, the statement deletes only the row on which the cursor is positioned — this is a **positioned delete**

►► See 3.3, “Using a cursor” on page 3-12, for information about positioned deletes.

**Searched deletes:** The statement in this example deletes all rows from the BENEFITS table for a fiscal year that precedes the one specified in the :FISCAL-YEAR host variable:

```
EXEC SQL
  DELETE FROM BENEFITS
    WHERE FISCAL_YEAR < :FISCAL-YEAR
END-EXEC.
```

If no rows satisfy the selection criteria in the WHERE clause of a searched delete, SQLCODE will be set to 100.

**Checking DELETE status:** The DBMS disallows an attempt to delete a row from a referenced table in a relationship if a row with a matching foreign key exists in a referencing table.

For example, since a referential constraint has been created between the EMPLOYEE table and the POSITION table (with column EMP\_ID in POSITION referencing column EMP\_ID in EMPLOYEE), you cannot delete employee 1234 from the EMPLOYEE table if employee 1234 exists in the POSITION table.

To detect a referential constraint violation on a DELETE statement, test for SQLCERC = 1060:

```
IF SQLCERC = 1060 PERFORM REFERENTIAL-ERROR
ELSE IF SQLCODE < 0 GOTO ERROR-ROUTINE.
```

**Automatic rollback:** If the attempt to delete one row of a searched delete fails:

- Statement execution halts
- The DBMS returns an error value to SQLCODE

- The results of the DELETE statement are automatically rolled back

### Caution using DELETE!:

#### CAUTION:

When you issue a delete, be sure that the statement includes a WHERE clause. If the WHERE clause is omitted, CA-IDMS deletes *all* rows from the named table.

## 3.2.5 Using indicator variables in data manipulation

**Indicator variables in SELECT or FETCH:** When a column value is retrieved into a host variable that has an associated indicator variable, the DBMS assigns a value to the indicator variable:

Indicator variable value	Meaning
-1	The value assigned to the host variable was null. The actual content of the host variable is unpredictable.
0	The host variable contains a non-null value that has not been truncated.
1 or greater	The host variable contains a truncated value. The value in the indicator variable is the length in bytes of the original untruncated value.

**Retrieving a null value:** Since a null value is not valid in the program language, the program must test for -1 in the indicator variable and direct processing to handle null value retrieval as needed if the test is true.

**Null retrieval example:** In the example below, the program initializes two numeric host variables to zero:

```
MOVE ZERO TO VAC-TAKEN.  
MOVE ZERO TO SICK-TAKEN.
```

If the statement below now retrieves null values from the VAC\_TAKEN and SICK\_TAKEN columns, the value of VAC-TAKEN and SICK-TAKEN are still zero because the actual content of the host variables is unchanged when nulls are retrieved:

```
EXEC SQL  
  SELECT VAC_TAKEN,  
        SICK_TAKEN  
    INTO :VAC-TAKEN INDICATOR :VAC-TAKEN-I,  
         :SICK-TAKEN INDICATOR :SICK-TAKEN-I  
   FROM BENEFITS  
 WHERE EMP_ID = :EMP-ID  
END-EXEC.
```

**Indicator variables in inserts and updates:** When the program issues a statement to store a value contained in a host variable, the statement optionally can name the associated indicator variable.

If the statement names the indicator and the indicator variable value is 0, the DBMS stores the actual content of the host variable. If the indicator variable value is -1, the DBMS stores a null value instead of the actual content of the host variable.

**Update examples with indicator variables:** In the example below, the program assigns 0 to the indicator variable after changing the value of the host variable VAC-TAKEN. This means CA-IDMS will store the actual content of VAC-TAKEN on the subsequent update:

```
ADD INPUT-VAC-TAKEN TO VAC-TAKEN.  
MOVE ZERO TO VAC-TAKEN-I.  
. . .  
EXEC SQL  
    UPDATE BENEFITS  
        SET VAC_TAKEN = :VAC-TAKEN INDICATOR :VAC-TAKEN-I  
        WHERE EMP_ID = :EMP-ID  
END-EXEC.
```

By omitting reference to the indicator variable in the UPDATE statement, the program can achieve the same result of storing the actual content of the host variable:

```
ADD INPUT-VAC-TAKEN TO VAC-TAKEN.  
. . .  
EXEC SQL  
    UPDATE BENEFITS  
        SET VAC_TAKEN = :VAC-TAKEN  
        WHERE EMP_ID = :EMP-ID  
END-EXEC.
```

Similarly, the program can store a null value without naming the indicator variable:

```
EXEC SQL  
    UPDATE BENEFITS  
        SET VAC_TAKEN = NULL  
        WHERE EMP_ID = :EMP-ID  
END-EXEC.
```

## 3.3 Using a cursor

In application programming, a cursor is an SQL construct that the program uses to process data in a result table. The cursor declaration defines the result table. Once the program declares the cursor, the program can open the cursor and sequentially fetch one row at a time from the result table.

### 3.3.1 Declaring a cursor

**How you declare a cursor:** You define a cursor by issuing a DECLARE CURSOR statement. The DECLARE CURSOR statement contains a SELECT statement:

```
EXEC SQL
    DECLARE EMP_SUM CURSOR FOR
        SELECT EMP_ID,
               MANAGER_ID,
               EMP_FNAME,
               EMP_LNAME,
               DEPT_ID
        FROM EMPLOYEE
        ORDER BY DEPT_ID
END-EXEC.
```

**Updateable cursors:** If the program updates the current cursor row, the cursor declaration must contain the FOR UPDATE OF clause, specifying the result table columns that may be updated. In the definition of an updateable cursor:

- Only one table is named in the FROM clause of the SELECT statement
- The named table must be a base table, an updateable view or a table procedure
- The outer select may not contain a UNION, ORDER BY, or GROUP BY clause

►► Refer to the documentation of the DECLARE CURSOR statement in the *CA-IDMS SQL Reference Guide* for all criteria that an updateable cursor must meet.

**Updateable cursor declaration example:** In this example, the EMP\_SUM cursor is declared to allow the program to update the MANAGER\_ID and DEPT\_ID columns:

```
EXEC SQL
    DECLARE EMP_SUM CURSOR FOR
        SELECT EMP_ID,
               MANAGER_ID,
               EMP_FNAME,
               EMP_LNAME,
               DEPT_ID
        FROM EMPLOYEE
        FOR UPDATE OF MANAGER_ID,
                   DEPT_ID
END-EXEC.
```

### 3.3.2 Fetching a row

**Opening the cursor:** Before the program can fetch cursor rows, it must open the cursor with an OPEN statement:

```
EXEC SQL  
  OPEN EMP_SUM  
END-EXEC.
```

**How you fetch a row:** The program fetches a row with a FETCH statement that names the cursor and includes an INTO clause that specifies the host variables to receive the fetched row:

```
EXEC SQL  
  FETCH EMP_SUM  
    INTO :EMP-ID,  
         :MANAGER-ID-I,  
         :EMP-FNAME,  
         :EMP-LNAME,  
         :DEPT-ID  
END-EXEC.
```

**Cursor position:** Cursor position refers to a current position relative to a row of the cursor. When a FETCH statement is executed, the values assigned to the host variables are retrieved from the row that follows the cursor position.

When the program opens the cursor, cursor position is before the first row of the result table. When a row is fetched, the cursor position moves to that row and the column values for that row are moved into the host variables.

If another FETCH statement is executed while the cursor remains open, cursor position moves to the next row.

**When there are no more rows:** Cursor position advances row by row with each FETCH. If there is no row following the cursor position and a FETCH statement is executed, the DBMS returns 100 to SQLCODE. When this condition occurs, the program should end iterative logic for fetching cursor rows.

**Testing for no more cursor rows:** To test for no more cursor rows, test for SQLCODE = 100. If the test result is true, set a variable to indicate this condition, as shown in the use of END-FETCH in the example below.

Referencing a variable such as END-FETCH in subsequent program logic is recommended because the program controls the variable value, whereas the DBMS controls the value of SQLCODE.

```
WORKING-STORAGE SECTION.  
 77 END-FETCH    PIC X VALUE 'N'.  
. . .  
PROCEDURE DIVISION.  
. . .  
. . .  
***** Perform paragraph until no more cursor rows to process  
PERFORM FETCH-CURSOR UNTIL END-FETCH = Y.  
. . .  
FETCH-CURSOR.  
  
  EXEC SQL  
    FETCH EMP_SUM INTO  
      EMP-ID,  
      MANAGER-ID MANAGER-ID-I,  
      EMP-FNAME,  
      EMP-LNAME,  
      DEPT-ID  
  END-EXEC.  
  
***** Test for no more cursor rows  
  IF SQLCODE = 100 MOVE 'Y' TO END-FETCH.  
. . .
```

**Closing a cursor:** The program can close a cursor with the CLOSE statement:

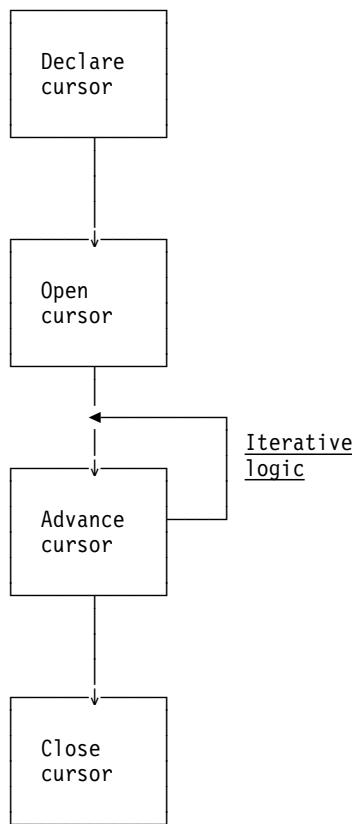
```
EXEC SQL  
  CLOSE EMP_SUM  
END-EXEC.
```

**Automatic closing of a cursor:** The COMMIT and ROLLBACK statements automatically close all open cursors used by the application program.

**Invalid cursor state:** The DBMS returns an *invalid cursor state* condition and ignores the statement if the program issues:

- An OPEN statement for a cursor that is open
- A CLOSE statement for a cursor that is closed
- A FETCH statement for a cursor that is closed
- A FETCH statement when the cursor position is after the last row (which means that the DBMS already returned 100 to SQLCODE)

**Summary of cursor management:** This diagram summarizes how the program uses a cursor:



### 3.3.3 Executing a positioned update or delete

**About positioned updates:** A positioned update modifies one or more column values of the current row of an updateable cursor. The statement takes this form:

```

EXEC SQL
UPDATE table-name
  SET column-name = value-specification
  ...
  WHERE CURRENT OF cursor-name
END-EXEC.
  
```

**Requirements for a positioned update:** To execute a positioned update, the program must declare a cursor that:

- Is updateable
- Contains a FOR UPDATE OF clause

**Advantage of an updateable cursor:** When the database transaction running under cursor stability fetches a row from an updateable cursor, the DBMS places a lock on the row and maintains it until one of these events occurs:

- The program fetches the next cursor row
- The cursor is closed

- The database transaction ends

In this way, CA-IDMS guarantees the base row is not modified or deleted by another transaction while it is the current cursor row.

The DBMS maintains the lock on the current row of an updateable cursor during a suspended SQL session. This feature is designed for pseudoconversational programming.

►► See 6.3, “Pseudoconversational programming” on page 6-9, for information about pseudoconversational programming with embedded SQL.

**Checking positioned update status:** If the program attempts to execute a positioned update when the referenced cursor is not updateable or does not contain a FOR UPDATE OF clause, the DBMS returns an *invalid cursor state* error.

►► See 3.2.3, “Modifying data” on page 3-7, for details about checking the status of UPDATE statements in general.

**Positioned update example:** In this example, the program declares a cursor to retrieve current data for vacation and sick days taken by employees. The program adds input values to the values retrieved for the employee in the current cursor row. Then the program issues a positioned update.

```

EXEC SQL
DECLARE VAC_SICK_CURSOR CURSOR FOR
  SELECT EMP_ID,
         VAC_TAKEN,
         SICK_TAKEN
    FROM BENEFITS
   FOR UPDATE OF VAC_TAKEN,
              SICK_TAKEN
END-EXEC.
.
.

EXEC SQL
  OPEN VAC_SICK_CURSOR
END-EXEC.
.
.

EXEC SQL
  FETCH VAC_SICK_CURSOR INTO
    :EMP-ID,
    :VAC-TAKEN INDICATOR VAC-TAKEN-I,
    :SICK-TAKEN INDICATOR SICK-TAKEN-I
END-EXEC.
.
.

ADD INPUT-VAC-TAKEN TO VAC-TAKEN
ADD INPUT-SICK-TAKEN TO SICK-TAKEN
.
.

EXEC SQL
  UPDATE BENEFITS
    SET VAC_TAKEN = :VAC-TAKEN,
        SICK_TAKEN = :SICK-TAKEN
   WHERE CURRENT OF VAC-SICK-CURSOR
END-EXEC.
.
.

EXEC SQL
  CLOSE VAC_SICK_CURSOR
END-EXEC.

```

**Positioned deletes:** You can delete the current row of an updateable cursor simply by naming the table and the cursor in the DELETE statement:

`DELETE FROM table-name WHERE CURRENT OF cursor-name`

A cursor must be updateable to perform a positioned delete, but the FOR UPDATE OF clause is not required in the cursor declaration.

**Checking positioned delete status:** If the program attempts to execute a positioned delete when the referenced cursor is not updateable, the DBMS returns an *invalid cursor state* error.

► See 3.2.4, “Deleting data” on page 3-9, for details about checking the status of DELETE statements in general.

**Positioned delete example:** In this example, the program declares an updateable cursor. After fetching a row, the program conditionally executes a positioned delete.

```
EXEC SQL
  DECLARE DEL_POSITION CURSOR FOR
    SELECT EMP_ID,
           JOB_ID
      FROM POSITION
END-EXEC.

.
.

EXEC SQL
  OPEN DEL_POSITION
END-EXEC.

.
.

EXEC SQL
  FETCH DEL_POSITION INTO
    :EMP-ID,
    :JOB-ID
END-EXEC.

.
.

IF INPUT-ACTION = 'D&rq
EXEC SQL
  DELETE FROM POSITION
    WHERE CURRENT OF DEL_POSITION
END-EXEC.

.
.

EXEC SQL
  CLOSE DEL_POSITION
END-EXEC.
```

## 3.4 Bulk processing

A CA-IDMS extension of ANSI-standard SQL allows you to transfer multiple rows of data between the database and the program using a single SELECT, FETCH, or INSERT statement with a BULK clause.

To issue a bulk select, fetch, or insert, the program must declare a host variable array.

►► See 4.2.2, “Defining host variables” on page 4-6, for information about declaring a host variable array in CA-ADS See 4.3.2, “Defining host variables” on page 4-14, for information about declaring a host variable array in COBOL. See 4.4.2, “Defining host variables” on page 4-30, for information about declaring a host variable array in PL/I.

### 3.4.1 Executing a bulk fetch

**About bulk fetch:** A bulk fetch is a FETCH statement that retrieves multiple rows from a cursor into a host variable array.

To execute a bulk fetch:

1. Declare a host variable array
2. Open the cursor
3. Issue a FETCH statement with the BULK clause

►► Refer to the *CA-IDMS SQL Reference Guide* for documentation of the FETCH statement.

**Cursor position:** The first execution of a FETCH BULK statement retrieves the first set of rows from the cursor result table. After statement execution, cursor position is on the last row fetched. If the FETCH BULK statement is executed again before the cursor is closed, the next set of rows retrieved begins with the row following the cursor position. Fetching proceeds sequentially through the cursor result table until no more rows are found.

**How many rows are fetched?:** If you do not specify a ROWS parameter in the BULK clause, the FETCH statement will retrieve as many rows as will fit between the starting row of the array and the end of the array.

If you specify a ROWS parameter in the BULK clause, the FETCH statement will retrieve a number of rows equal to the value in the ROWS. This value must be less than or equal to the number of rows between the starting row of the array and the end of the array.

**Maximum rows example:** In this example, the program assigns a ROWS value that corresponds to the number of rows that can be displayed on a given display terminal:

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 BULK-DIVISION.
  02 BULK-DIV OCCURS 100 TIMES.
    03 DEPT-ID      PIC 9(4).
    03 DEPT-NAME    PIC X(40).
  01 DIV-CODE      PIC X(3).
  01 WS-SCREEN-LENGTH  PIC S9(4) COMP.
  .
  .
  EXEC SQL
    DECLARE DIV_DEPT CURSOR FOR
      SELECT DEPT_ID, DEPT_NAME
        FROM DEPARTMENT
          WHERE DIV_CODE = :DIV-CODE
END-EXEC.
ACCEPT SCREENSIZE INTO WS-SCREEN-LENGTH.
SUBTRACT 4 FROM WS-SCREEN-LENGTH.
IF WS-SCREEN-LENGTH > 100 MOVE 100 TO
WS-SCREEN LENGTH.
  .
  .
  MOVE INPUT-DIV-CODE TO DIV-CODE.

EXEC SQL
OPEN DIV_DEPT
END-EXEC.

FETCH-PARAGRAPH.

EXEC SQL
  FETCH DIV_DEPT
    BULK :BULK-DIVISION ROWS :WS-SCREEN-LENGTH
END-EXEC.

IF SQLCODE=100 MOVE 'Y' TO END-FETCH.
  .
  .
  .
  (Iterate paragraph until no more rows)

```

**Specifying a starting row:** The DBMS assigns the first row of the result table to the first row of the array unless you include the START parameter on the BULK clause. The START value corresponds to the subscript value of the array occurrence.

**Checking statement execution:** If program logic calls for repeating the FETCH BULK statement until no more rows are found, the program must test for SQLCODE = 100, as described in 3.3, “Using a cursor” on page 3-12. The DBMS always sets the value of SQLCNRP equal to the number of rows returned unless an error occurs during processing.

The table below shows the possible combination of values returned to SQLCODE and SQLCNRP on a FETCH BULK statement:

Result of bulk fetch	SQLCODE value	SQLCNRP value
No rows are returned	100	0
At least one row is returned but fewer rows than the maximum allowed	100	Equals the number of rows returned
The number of rows returned matches the maximum allowed	0	Equals the number of rows returned

**Advantages of a bulk fetch:** Using a BULK clause with a FETCH statement minimizes resources to retrieve data.

Unlike a bulk select, the program can retrieve an unlimited number of result rows by repeating a bulk fetch.

### Bulk fetch considerations

- With a bulk fetch, the program generally cannot perform current or cursor operations such as a positioned update or delete because the cursor is always positioned on (or after) the last row fetched
- If an error occurs during the processing of a bulk fetch, the contents of the host variable array are unpredictable
- If a bulk fetch results in retrieval of a null value, the contents of the host variable for the corresponding column is unpredictable

**Bulk fetch example:** In this example, the program issues an INCLUDE TABLE statement to declare a host variable array for several columns of the EMPLOYEE table. Then it declares a cursor to select the column values from all rows of the table.

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
EXEC SQL
INCLUDE TABLE EMPLOYEE AS BULK-EMPLOYEE
(EMP_ID, EMP_FNAME, EMP_LNAME, DEPT_ID)
NUMBER OF ROWS 50
PREFIX 'BULK-'
END-EXEC.
EXEC SQL END DECLARE SECTION END-EXEC.
.
.
.
EXEC SQL
DECLARE EMP_CRSR CURSOR FOR
SELECT EMP_ID,
       EMP_FNAME,
       EMP_LNAME,
       DEPT_ID
      ORDER BY 4, 3, 2
END-EXEC.

```

When the FETCH statement is executed, the first 50 rows of the cursor result table are assigned to the BULK-EMPLOYEE array, because the default starting row assignment is 1 and the default number of rows assigned is the array size. If the FETCH statement is repeated, the next 50 rows of the result table are assigned to the array.

```
EXEC SQL
  OPEN EMP_CRSR
END-EXEC.
.
.
.
EXEC SQL
  FETCH EMP_CRSR
  BULK :BULK-EMPLOYEE
END-EXEC.

IF SQLCODE = 100 MOVE 'Y' TO END-FETCH.
```

### 3.4.2 Executing a bulk select

**About bulk select:** A bulk select is a SELECT statement that retrieves multiple rows from the database into a host variable array:

1. Declare a host variable array
2. Issue the SELECT statement with a BULK clause, as in this example:

```
EXEC SQL
  SELECT DEPT_ID,
         DEPT_NAME,
         DIV_CODE,
         DEPT_HEAD_ID
    BULK :BULK-DEPARTMENT
   FROM DEPARTMENT
END-EXEC.
```

**Checking the status of a bulk select:** A successful bulk select returns 100 to SQLCODE. A value of 100 will be returned if there are fewer result rows than entries in the bulk array or if the number of result rows is the same as the number of entries. If the array is too small for the result table, the statement returns a *cardinality violation* error.

The table below shows the possible combinations of SQLCODE and SQLCNRP values on a bulk select:

Result of bulk select	SQLCODE value	SQLCNRP value
No rows are returned	100	0
At least one row is returned but fewer rows than the maximum allowed	100	Greater than 0 and less than or equal to the maximum allowed
The number of rows returned exceeds the maximum allowed	Less than 0	Equal to the maximum allowed

**Advantage of a bulk select:** A bulk select retrieves a set of rows using fewer resources than a series of single-row SELECT statements to retrieve the same rows.

**Bulk select considerations:** A bulk select:

- Cannot retrieve more rows than there are occurrences in the host variable array
- Retrieves the same set of rows, not the next set of rows, if the statement is reissued within the database transaction
- Causes the contents of the host variable array to be unpredictable if an error occurs during processing

This means that a bulk select is appropriate only when selecting from a table with a number of rows that you consider fixed, such as a table of the 50 states and their mailing codes.

If the size of the host variable array may be too small for the result table, you should declare a cursor for the SELECT statement and use a bulk fetch.

### 3.4.3 Executing a bulk insert

**About bulk insert:** a bulk insert is an INSERT statement that adds multiple rows in a host variable array to the database.

To execute a bulk insert:

1. Declare a host variable array
2. Assign values to the host variable array
3. Issue the INSERT statement with the BULK clause

**Specifying the START and ROWS parameters:** A bulk insert adds as many rows from the host variable array as are specified in the ROWS parameter, starting from the row specified in the START parameter. If START and ROWS are not specified, these are the defaults:

- The starting row is the first entry in the array
- The number of rows inserted is the number of occurrences defined for the array

**Note:** If the array is not full, specify a ROWS parameter value equal to the number of occurrences in the array that contain data. This ensures that the DBMS will not attempt to insert array occurrences that contain no data.

**Bulk insert example:** In this example, the program declares a host-variable array with an INCLUDE TABLE statement. After values are assigned to the array, the program issues a statement to insert all of the data in the array:

```

EXEC SQL
  INCLUDE TABLE SKILL AS BULK-SKILL
    NUMBER OF ROWS 100
    PREFIX 'BULK-'
END-EXEC.

.
.

(Assign values to BULK-SKILL array)
.

.

EXEC SQL
  INSERT INTO SKILL
    BULK :BULK-SKILL
      ROWS :NUM-ROWS
END-EXEC.

IF SQLCODE < 0
  MOVE SQLCNRP TO FAILING-ROW-NUM
  PERFORM ERROR-ROUTINE.

```

**Checking bulk insert status:** To detect unsuccessful execution of a bulk insert, test for SQLCODE < 0.

If the result of the test is true, the value of SQLCNRP equals the relative row number (from the specified starting row) of the row which caused the failure. The DBMS rolls back the results of the failing row but not the results of the prior rows.

The table below shows the possible combinations of SQLCODE and SQLCNRP values on a bulk insert:

Result of bulk insert	SQLCODE value	SQLCNRP value
Fewer rows than the number of rows specified are inserted because the insert failed on a row	Less than 0	Equal to the relative row number of the failing row
The number of rows inserted matches the number of rows specified	100	Equal to the number of rows inserted

**Advantage of a bulk insert:** A bulk insert adds a group of rows using fewer resources than if the program issues a separate INSERT statement for each row.

## 3.5 Invoking external routines

### 3.5.1 Call Statement

In a program, you can use the CALL statement to invoke an external routine. The external routine can be an SQL procedure or an SQL table procedure.

► Refer to the *CA-IDMS SQL Reference Guide* for more information on SQL procedures and table procedures.

#### 3.5.1.1 CALL of a procedure

A procedure always returns zero or one result sets of parameters.

```
EXEC SQL
  CALL DEMOEMPL.GET_BONUS
    (1234, :BONUS-AMOUNT, :BONUS-CURRENCY)
END-EXEC
```

If the CALL is successful, indicated by an SQLSTATE of '00000' the host variables BONUS-AMOUNT and BONUS-CURRENCY will contain valid data, returned by the invoked routine for EMP-ID 1234, the input value supplied for the first parameter.

#### 3.5.1.2 CALL of a table procedure

A table procedure can return zero or more result sets of parameters. Therefore, a simple CALL statement can not be used to invoke and return all the result sets of the table procedure; a cursor is required.

##### Declaration of the cursor:

```
EXEC SQL
  DECLARE C_BONUS_SET CURSOR
    FOR CALL DEMOEMPL.GET_BONUS_SET
      (EMP_ID =1234 )
END-EXEC.
```

##### Opening the cursor:

```
EXEC SQL
  OPEN C_BONUS_SET
END-EXEC.
```

##### Fetching the result sets:

```
EXEC SQL
  FETCH C_BONUS_SET INTO
    :EMP-ID,
    :BONUS-AMOUNT,
    :BONUS-CURRENCY
END-EXEC.
```

Host variables for all parameters specified in the table procedure definition should be provided.

►► See 3.3, “Using a cursor” on page 3-12 for further information about using cursors.

## 3.5.2 SELECT statement

The SELECT statement may be used as an alternative to the CALL statement to invoke an external routine. Again, the external routine can be an SQL procedure or an SQL table procedure.

### 3.5.2.1 SELECT of a procedure

A procedure always returns zero or one result sets of parameters, therefore, a SELECT ... INTO is used.

```
EXEC SQL
    SELECT BONUS_AMOUNT,
           BONUS_CURRENCY
    FROM DEMOEMPL.GET_BONUS(1234)
    INTO :BONUS-AMOUNT,
         :BONUS_CURRENCY
END-EXEC
```

If the SELECT is successful, indicated by an SQLSTATE of '00000' the host variables BONUS-AMOUNT and BONUS-CURRENCY will contain valid data, returned by the invoked routine for EMP-ID 1234, the input value supplied for the first parameter.

### 3.5.2.2 SELECT of a table procedure

A table procedure can return zero or more result sets of parameters. Therefore, a SELECT ... INTO statement is only used when the SELECT returns zero or only one result set. A cursor is required if more than one row is returned to the result set.

#### Declaration of the cursor:

```
EXEC SQL
    DECLARE C_BONUS_SET CURSOR
        FOR SELECT BONUS_AMOUNT, BONUS_CURRENCY
        FROM DEMOEMPL.GET_BONUS_SET
        ( EMP_ID =1234 )
END-EXEC.
```

#### Opening the cursor:

```
EXEC SQL
    OPEN C_BONUS_SET
END-EXEC.
```

#### Fetching the result sets:

```
EXEC SQL
  FETCH C_BONUS_SET
  INTO :BONUS-AMOUNT,
       :BONUS-CURRENCY
END-EXEC.
```

►► See 3.3, “Using a cursor” on page 3-12 for further information about using cursors.



# Chapter 4. Requirements and Options for Host Languages

---

4.1	About this chapter	4-3
4.2	Using SQL in a CA-ADS application	4-4
4.2.1	Embedding SQL statements	4-4
4.2.1.1	Delimited, continued, and commented statements	4-4
4.2.1.2	Placing an SQL statement	4-5
4.2.2	Defining host variables	4-6
4.2.2.1	Including tables	4-8
4.2.2.2	Defining bulk structures	4-9
4.2.3	Referring to host variables	4-10
4.2.4	Including SQL Communication Areas	4-10
4.3	Using SQL in a COBOL application program	4-12
4.3.1	Embedding SQL statements	4-12
4.3.1.1	Delimited, continued, and commented statements	4-12
4.3.1.2	Placing an SQL statement	4-13
4.3.2	Defining host variables	4-14
4.3.2.1	Using COBOL data declarations	4-14
4.3.2.2	Using INCLUDE TABLE	4-17
4.3.2.3	Defining bulk structures	4-19
4.3.2.4	Non-bulk structures and indicator arrays	4-20
4.3.3	Referring to host variables	4-22
4.3.4	Including SQL Communication Areas	4-23
4.3.5	Copying information from the dictionary	4-24
4.3.6	COPY IDMS FILE statement	4-25
4.3.6.1	Purpose	4-25
4.3.6.2	Syntax	4-25
4.3.6.3	Parameters	4-25
4.3.6.4	Usage	4-25
4.3.7	COPY IDMS RECORD statement	4-25
4.3.7.1	Purpose	4-25
4.3.7.2	Syntax	4-26
4.3.7.3	Parameters	4-26
4.3.7.4	Usage	4-26
4.3.8	COPY IDMS MODULE statement	4-27
4.3.8.1	Purpose	4-27
4.3.8.2	Syntax	4-27
4.3.8.3	Parameters	4-27
4.3.8.4	Usage	4-27
4.3.9	INCLUDE module-name statement	4-28
4.3.10	Non-SQL precompiler directives	4-28
4.4	Using SQL in a PL/I application program	4-29
4.4.1	Embedding SQL statements	4-29
4.4.1.1	Declaring SQLXQ1	4-29
4.4.1.2	Delimited, continued, and commented statements	4-29
4.4.2	Defining host variables	4-30
4.4.2.1	Using PL/I declarations	4-30

---

4.4.2.2 Using INCLUDE TABLE . . . . .	4-32
4.4.2.3 Defining bulk structures . . . . .	4-34
4.4.3 Referring to host variables . . . . .	4-35
4.4.4 Including SQL Communication Areas . . . . .	4-36
4.4.5 Including information from the dictionary . . . . .	4-37
4.4.6 INCLUDE IDMS record statement . . . . .	4-38
4.4.6.1 Purpose . . . . .	4-38
4.4.6.2 Syntax . . . . .	4-38
4.4.6.3 Parameters . . . . .	4-38
4.4.6.4 Usage . . . . .	4-39
4.4.7 INCLUDE IDMS MODULE statement . . . . .	4-39
4.4.7.1 Purpose . . . . .	4-39
4.4.7.2 Syntax . . . . .	4-39
4.4.7.3 Parameters . . . . .	4-39
4.4.8 INCLUDE module-name statement . . . . .	4-40
4.4.9 Non-SQL precompiler directives . . . . .	4-40

## 4.1 About this chapter

This chapter describes requirements and options that apply to a particular host language when you embed SQL in an application program to access CA-IDMS. This chapter discusses each of the following languages:

- CA-ADS
- COBOL
- PL/I

Topics for each language-specific section include:

- Formatting SQL statements
- Declaring host variables
- Supported data types

## 4.2 Using SQL in a CA-ADS application

This section presents information that is specific to embedding SQL in a CA-ADS application program.

**Note:** Refer to the following manuals for documentation of all aspects of CA-ADS application programming:

- *CA-ADS User Guide*
- *CA-ADS Reference Guide*

### 4.2.1 Embedding SQL statements

**Requirements:** To embed an SQL statement in a CA-ADS program, you must:

- Observe CA-ADS margin requirements (columns 1 to 72)
- Use SQL statement delimiters

**Options:** You can use the SQL convention to insert comments in an SQL statement.

You can use the CA-ADS convention to continue an SQL statement on the next line.

#### 4.2.1.1 Delimited, continued, and commented statements

**How you delimit a statement:** When you embed an SQL statement in a CA-ADS application program, you must use these statement delimiters:

- Begin each SQL statement with **EXEC SQL**
- End each SQL statement with **END-EXEC.**

**Statement delimiter example:** The example below shows the use of SQL statement delimiters:

```
EXEC SQL
  INSERT INTO DIVISION VALUES ('D07','LEGAL',1234)
END-EXEC.
```

The statement text can be on the same line as the delimiters.

**Continuing statements:** You can write an SQL statement on more than one line if you do one of the following:

- Split the statement before or after any keyword, value, or delimiter
- Code through column 72 of one line and continue in column 1 of the next line

#### Continued statement example

```
-----1-----2-----3-----4-----5-----6-----7--  
EXEC SQL  
    INSERT INTO SKILL VALUES (5678, 'TELEMARKETING', 'PRESENT SALES SCRIP  
T OVER THE TELEPHONE, INPUT RESULTS')  
END-EXEC.
```

**How to put comments in SQL statements:** To include comments within SQL statements embedded in a CA-ADS program, you can use the SQL comment characters, two consecutive hyphens (--), on an SQL statement line following the statement text.

### Restrictions on comments

- Don't insert a comment in the middle of a string constant or delimited identifier
- Don't use the CA-ADS comment character ! to insert a comment in an embedded SQL statement

**SQL comment example:** The example below shows two comments within an embedded SQL statement:

```
EXEC SQL  
-- Perform update on active employees only  
UPDATE BENEFITS  
    SET VAC_ACCRUED = VAC_ACCRUED + 10,      -- Add 10 hours vacation  
        SICK_ACCRUED = SICK_ACCRUED + 1      -- Add 1 sick day  
    WHERE EMP_ID IN  
        (SELECT EMP_ID FROM EMPLOYEE  
         WHERE STATUS = 'A')  
END-EXEC.
```

#### 4.2.1.2 Placing an SQL statement

**Where you can put statements:** These are the rules for placing an SQL statement in a CA-ADS program:

- Only a WHENEVER directive or a DECLARE CURSOR statement may appear in a declaration module
- All SQL statements *except* for INCLUDE TABLE are valid for premap and response processes

**Order of compilation:** Dialog modules are compiled in this order:

1. Declaration module
2. Premap process module
3. Response process modules

The order of compilation of response process modules is not guaranteed. Therefore, if a WHENEVER condition or the availability of a cursor must span modules, you should place the WHENEVER statement or cursor declaration in a declaration module.

**Declaration module:** CA-ADS uses a declaration module, if it exists, when you compile the dialog.

The declaration module can contain WHENEVER directives and DECLARE CURSOR statements.

WHENEVER and DECLARE CURSOR are not executable statements, and a declaration module is not executable. The scope of a WHENEVER or DECLARE CURSOR is the entire dialog.

A WHENEVER directive or DECLARE CURSOR statement is valid in a premap or response process, but the scope of the statement is not global.

**Scope of WHENEVER:** The scope of a WHENEVER condition in a premap or response is the rest of that premap or response or until another WHENEVER statement that changes the condition is encountered within the process.

A WHENEVER declaration in a premap or response overrides (for the duration of its scope) the global declaration in the declaration module.

**Scope of DECLARE CURSOR:** The scope of a DECLARE CURSOR statement is from the moment that the declaration is encountered in dialog compilation to the end of that compilation.

## 4.2.2 Defining host variables

**What you declare:** You implicitly declare host variables for a CA-ADS dialog when:

- You associate a record or a table with the dialog using the WORK RECORD screen of ADSC
- You associate a map or subschema, and thus its records, with the dialog

Any record element that is valid for a CA-ADS MOVE command is valid as a host variable.

►► Refer to *CA-ADS Reference Guide* for information about ADSC and the MOVE command.

**Equivalent column data types:** All CA-IDMS data types are supported by CA-ADS.

This table shows definitions of CA-ADS host variable data types and the equivalent CA-IDMS table column data types:

<b>PICTURE and USAGE clause</b>	<b>CA-IDMS data type</b>
PIC X( <i>n</i> ) USAGE DISPLAY	CHAR( <i>n</i> )
01 <u>name</u> 49 <u>name</u> -LEN PIC S9(4) COMP 49 <u>name</u> -TEXT PIC X( <i>n</i> )	VARCHAR( <i>n</i> )
PIC S9( <i>p-s</i> )V9( <i>s</i> ) USAGE COMP-3	DECIMAL( <i>p,s</i> )
PIC 9( <i>p-s</i> )V9( <i>s</i> ) USAGE COMP-3	UNSIGNED DECIMAL( <i>p,s</i> ) <sup>1</sup>
USAGE COMP-2	DOUBLE PRECISION
USAGE COMP-1	REAL
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> <5)	SMALLINT
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> >4 and <u>n</u> <10)	INTEGER
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> >9)	LONGINT <sup>1</sup>
PIC S9( <i>p-s</i> )V9( <i>s</i> ) USAGE DISPLAY	NUMERIC( <i>p,s</i> )
PIC 9( <i>p-s</i> )V9( <i>s</i> ) USAGE DISPLAY	UNSIGNED NUMERIC( <i>p,s</i> ) <sup>1</sup>
PIC X( <i>n</i> ) USAGE DISPLAY	BINARY( <i>n</i> ) <sup>1</sup>
PIC G( <i>n</i> ) USAGE DISPLAY-1	GRAPHIC( <i>n</i> ) <sup>1</sup>
01 <u>name</u> 49 <u>name</u> -LEN PIC S9(4) COMP 49 <u>name</u> -TEXT PIC G( <i>n</i> ) DISPLAY-1	VARGRAPHIC( <i>n</i> ) <sup>1</sup>
PIC X(10) USAGE DISPLAY	DATE <sup>1</sup>
PIC X(8) USAGE DISPLAY	TIME <sup>1</sup>
PIC X(26) USAGE DISPLAY	TIMESTAMP <sup>1</sup>

**Note:**

<sup>1</sup> This data type is a CA-IDMS extension of ANSI-standard SQL.

►► Refer to the *CA-IDMS SQL Reference Guide* for documentation of CA-IDMS data types.

### 4.2.2.1 Including tables

You include an SQL table in a CA-ADS dialog by specifying the table on the WORK RECORD screen of ADSC.

ADSC creates host variable structures using these data type equivalents when directed to include a table on the Work Record Screen:

CA-IDMS data type	Data type in included table
BINARY( <i>n</i> )	PIC X( <i>n</i> )
CHARACTER( <i>n</i> )	PIC X( <i>n</i> )
VARCHAR( <i>n</i> )	-LEN PIC S9(4) COMP -TEXT PIC X( <u>n</u> )
GRAPHIC( <i>n</i> )	PIC G( <i>n</i> ) DISPLAY-1
VARGRAPHIC( <i>n</i> )	-LEN PIC S9(4) COMP -TEXT PIC G( <u>n</u> ) DISPLAY-1
DECIMAL( <i>p,s</i> )	PIC S9( <i>p-s</i> )V9( <i>s</i> ) COMP-3
UNSIGNED DECIMAL( <i>p,s</i> )	PIC 9( <i>p-s</i> )V9( <i>s</i> ) COMP-3
NUMERIC( <i>p,s</i> )	PIC S9( <i>p-s</i> )V9( <i>s</i> ) DISPLAY
UNSIGNED NUMERIC( <i>p,s</i> )	PIC 9( <i>p-s</i> )V9( <i>s</i> ) DISPLAY
DOUBLE PRECISION	COMP-2
FLOAT( <u>n</u> ) where <u>n</u> <= 24 <u>n</u> > 24	COMP-1 COMP-2
REAL	COMP-1
DATE	PIC X(10)
TIME	PIC X(8)
TIMESTAMP	PIC X(26)
SMALLINT	PIC S9(4) COMP
INTEGER	PIC S9(8) COMP
LONGINT	PIC S9(18) COMP
<i>Indicator variable</i>	PIC S9(4) COMP or PIC S9(8) COMP

#### 4.2.2.2 Defining bulk structures

**About bulk structures:** A bulk structure is a group element or a record which contains a subordinate array for holding multiple occurrences of input or output values. Bulk structures are used in bulk SELECT, INSERT, and FETCH statements for retrieving or storing multiple rows of data.

**Format of a bulk structure:** A bulk structure consists of three levels:

- The highest level is the structure itself (level 01 through 47).
- The second level is a multiply occurring group item (level 02 through 48).
- The third level consists of elementary or variable length data items (variable length data items are group elements consisting of a halfword length field followed by a character or graphics field).

The number, type and order of data items at the lowest level must correspond to the number, data type, and order of column values being retrieved or inserted.

All data descriptions used by CA-ADS are defined within the dictionary.

**Bulk structure example:** The following is an example of a valid bulk structure definition using IDD syntax:

```
ADD ELEMENT EMP-ID PIC 999.  
ADD ELEMENT EMP-NAME PIC X(30).  
ADD ELEMENT DEPT-NAME PIC X(30).  
ADD ELEMENT BULK-ROW SUB ELEMENTS ARE  
    (EMP-ID EMP-NAME DEPT-NAME).  
ADD ELEMENT BULK-DATA SUB ELEMENT  
    BULK-ROW OCCURS 20.
```

**Referring to a bulk structure:** When referring to a bulk structure in a SELECT, FETCH, or INSERT statement, the name of the highest level is used:

```
EXEC SQL  
    FETCH EMPCURS BULK :BULK-DATA  
END-EXEC.
```

**Restrictions:** The following restrictions apply to bulk structures defined for use with CA-ADS:

- The following clauses may not appear within the lowest level element definitions:
  - BLANK WHEN ZERO IS ON
  - JUSTIFY IS ON
  - OCCURS
  - (R) indicating redefinition
  - SIGN IS LEADING/TRAILING
  - SYNC
- Indicator variables cannot be defined for elements within the bulk structure

- The bulk structure must be either a record or the first element within the record

### 4.2.3 Referring to host variables

**What you can do:** CA-IDMS supports references to host variables in SQL statements. The host variable name must be preceded with a colon (:).

►► See 2.2.3, “Referring to host variables” on page 2-7, for more information.

**Qualifying host variable names:** CA-IDMS supports two methods of qualifying CA-ADS host variable names in an SQL statement.

For example, assume these host variable definitions:

```
01 EMP
    03 HIRE-DATE
    .
    .
01 MGR
    03 HIRE-DATE
    .
    .
```

The methods of qualifying HIRE-DATE in both of the following examples are valid:

```
EXEC SQL
    SELECT...
        INTO :HIRE-DATE OF EMP
```

```
-----
```

```
EXEC SQL
    SELECT...
        INTO :EMP.HIRE-DATE
```

### 4.2.4 Including SQL Communication Areas

**Automatically included:** The SQL Communications Areas (SQLCAs) are included automatically in a CA-ADS dialog that contains embedded SQL. You make no declaration of these data structures in the CA-ADS modules you create.

**SQLCA structure:** This is the CA-ADS format of the SQLCA:

```

01 SQLCA.
  02 SQLCAID          PIC X(8).
  02 SQLCODE          PIC S9(8) COMP.
  02 SQLCSID          PIC X(8).
  02 SQLCINFO.
    03 SQLCERC          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCNRP          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCSER          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCLNO          PIC S9(8) COMP.
    03 SQLCMCT          PIC S9(8) COMP.
    03 SQLCARC          PIC S9(8) COMP.
    03 SQLCFJB          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
  02 SQLCINF2 REDEFINES SQLCINFO.
    03 SQLERRD          PIC S9(8) COMP
                        OCCURS 12.
  02 SQLCMMSG.
    03 SQLCERL          PIC S9(8) COMP.
    03 SQLERM           PIC X(256).
  02 SQLCMMSG2 REDEFINES SQLCMMSG.
    03 FILLER           PIC X(2).
    03 SQLERRM.
      04 SQLCERRML        PIC S9(4) COMP.
      04 SQLERRMRC        PIC X(256).
  02 SQLSTATE          PIC X(5).
  02 FILLER           PIC X(11).

  02 SQLWORK           PIC X(16).
  02 SQLWRK2 REDEFINES SQLWORK.
    03 SQLERRP.
      04 SQLCVAL          PIC X(5).
      04 FILLER           PIC X(3).
    03 SQLWARN.
      04 SQLWARN0          PIC X(1).
      04 SQLWARN1          PIC X(1).
      04 SQLWARN2          PIC X(1).
      04 SQLWARN3          PIC X(1).
      04 SQLWARN4          PIC X(1).
      04 SQLWARN5          PIC X(1).
      04 SQLWARN6          PIC X(1).
      04 SQLWARN7          PIC X(1). 
```

Included by the precompiler for DB2 compatibility; not used by CA-IDMS

## 4.3 Using SQL in a COBOL application program

This section presents information that is specific to embedding SQL in a COBOL application program.

**Note:** Refer to *CA-IDMS DML Reference - COBOL* for documentation of all aspects of COBOL application programming in the CA-IDMS environment.

### 4.3.1 Embedding SQL statements

**Requirements:** To embed an SQL statement in a COBOL program, you must:

- Place the statement in the proper division of the program
- Observe COBOL margin requirements (columns 8 to 72)
- Use statement delimiters

**Options:** You can use SQL conventions to:

- Continue an SQL statement on the next line
- Insert comments in an SQL statement

You can use a precompiler-directive statement to copy SQL statements in a module from the dictionary into the program.

**Note:** SQL statements cannot be embedded using the COBOL INCLUDE or BASIS statement.

#### 4.3.1.1 Delimited, continued, and commented statements

**Using SQL statement delimiters:** When you embed an SQL statement in a COBOL application program, you must use these statement delimiters:

- Begin each SQL statement with **EXEC SQL**
- End each SQL statement with **END-EXEC**.

**Note:** The period following END-EXEC is optional. Include it wherever you would normally terminate a COBOL statement with a period.

The example below shows the use of SQL statement delimiters:

```
EXEC SQL
    INSERT INTO DIVISION VALUES ('D07','LEGAL',1234)
END-EXEC.
```

The statement text can be on the same line as the delimiters.

**Continuing statements:** You can write SQL statements on one or more lines. No special character is required to show that a statement continues on the next line if you split the statement before or after any keyword, value, or delimiter.

You can use the COBOL continuation character, a hyphen (-), in column 7 when a string constant in an embedded SQL statement is split at column 72 and continued on the next line:

```
-----1-----2-----3-----4-----5-----6-----7--  
EXEC SQL  
    INSERT INTO SKILL  
        VALUES (5678,'TELEMARKETING','PRESENT SALES SCRIPT OVER THE  
-'ELEPHONE, INPUT RESULTS')  
END-EXEC.
```

**Inserting SQL comments:** To include comments within SQL statements embedded in a COBOL program, you can:

- Use the COBOL comment character \* in column 7
- Use the SQL comment characters, two consecutive hyphens (--), on an SQL statement line following the statement text

A comment that begins with the SQL comment characters (--) terminates at the end of the line (column 72).

You cannot use SQL comment characters to insert a comment in the middle of a string constant or delimited identifier.

The example below shows both methods of inserting comments within an embedded SQL statement:

```
-----1-----2-----3-----4-----5-----6-----7--  
EXEC SQL  
***** PERFORM UPDATE ON ACTIVE EMPLOYEES ONLY  
UPDATE BENEFITS  
    SET VAC_ACCRUED = VAC_ACCRUED + 10, -- Add 10 hours vacation  
        SICK_ACCRUED = SICK_ACCRUED + 1 -- Add 1 sick day  
    WHERE EMP_ID IN  
        (SELECT EMP_ID FROM EMPLOYEE  
        WHERE STATUS = 'A')  
END-EXEC.
```

### 4.3.1.2 Placing an SQL statement

**Where you can put statements:** These are the rules for placing an SQL statement in a COBOL program:

- The INCLUDE statement must be in the DATA DIVISION
- The WHENEVER can be in the DATA DIVISION or the PROCEDURE DIVISION
- The DECLARE CURSOR and DECLARE EXTERNAL CURSOR statements can be in the DATA DIVISION or the PROCEDURE DIVISION but must precede the OPEN statement that references the cursor
- All other statements must be in the PROCEDURE DIVISION

**Versions prior to VS COBOL 2:** If your program is written for a version of COBOL that is prior to VS COBOL 2, observe these guidelines:

- Do not include an SQL statement within the scope of a COBOL IF statement
- Use the THRU construction for a PERFORM statement that references a section containing an SQL statement

**COBOL version examples:** This example is valid in VS COBOL 2 and later versions:

```
IF I < 100
  EXEC SQL
    SELECT EMP_LNAME,
           DEPT_ID
      INTO :EMP-LNAME,
           :DEPT-ID
     WHERE EMP_ID = :WK-EMP-ID
  END-EXEC.
  COMPUTE A = A + 1.
```

For a version of COBOL prior to VS COBOL 2, the procedure above can be written:

```
IF I < 100
  PERFORM PARAGRAPH-B THRU PARAGRAPH-B-END
  COMPUTE A = A + 1.

PARAGRAPH-B.
  EXEC SQL
    SELECT EMP_LNAME,
           DEPT_ID
      INTO :EMP-LNAME,
           :DEPT-ID
     WHERE EMP_ID = :WK-EMP-ID
  END-EXEC.
PARAGRAPH-B-END.
```

## 4.3.2 Defining host variables

Host variables are defined using COBOL data declarative statements appearing in SQL declare sections.

CA-IDMS extensions offer the alternative methods of using the INCLUDE TABLE precompiler directive or copying record descriptions from the data dictionary.

A host variable definition may appear anywhere a legal data item definition can appear.

### 4.3.2.1 Using COBOL data declarations

**What you declare:** Within an SQL declare section, you specify the name, level, and data type of host variables using standard COBOL data declarative statements and observing these guidelines:

- A host variable **name** must conform to COBOL rules for forming variable names
- The level number is in the range of 01 to 49, or 77

A CA-IDMS extension of ANSI-standard SQL allows level numbers in the range of 02 to 49.

- The data type of the host variable as defined in the PICTURE and USAGE clauses

**Equivalent column data types:** All CA-IDMS data types can be supported in a COBOL program.

This table shows types of COBOL host variables and the equivalent CA-IDMS table column data types:

COBOL PICTURE and USAGE clause	CA-IDMS data type
PIC X(n) USAGE DISPLAY	CHAR(n)
01 <u>name</u> 49 <u>name</u> -LEN PIC S9(4) COMP 49 <u>name</u> -TEXT PIC X(n)	VARCHAR(n)
PIC S9(p-s)V9(s) USAGE COMP-3	DECIMAL(p,s)
PIC 9(p-s)V9(s) USAGE COMP-3	UNSIGNED DECIMAL(p,s) <sup>1</sup>
USAGE COMP-2	DOUBLE PRECISION
USAGE COMP-1	REAL
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> <5)	SMALLINT
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> >4 and <u>n</u> <10)	INTEGER
PIC S9( <u>n</u> ) USAGE COMP (where <u>n</u> >9)	LONGINT <sup>1</sup>
PIC S9(p-s)V9(s) USAGE DISPLAY	NUMERIC(p,s)
PIC 9(p-s)V9(s) USAGE DISPLAY	UNSIGNED NUMERIC(p,s) <sup>1</sup>
PIC X(n) USAGE SQLBIN	BINARY(n) <sup>1</sup>
PIC G(n) USAGE DISPLAY-1	GRAPHIC(n) <sup>1</sup>
01 <u>name</u> 49 <u>name</u> -LEN PIC S9(4) COMP 49 <u>name</u> -TEXT PIC G(n) DISPLAY-1	VARGRAPHIC(n) <sup>1</sup>
PIC X(10) USAGE DISPLAY	DATE <sup>1</sup>
PIC X(8) USAGE DISPLAY	TIME <sup>1</sup>
PIC X(26) USAGE DISPLAY	TIMESTAMP <sup>1</sup>

**Note:**

<sup>1</sup> This data type is a CA-IDMS extension of ANSI-standard SQL.

► Refer to the *CA-IDMS SQL Reference Guide* for documentation of CA-IDMS data types.

**Host variable declaration example:** In this example, the SQL declare section defines host variables, including one indicator variable, using standard COBOL data declarations. The example is annotated to show the equivalent column data type for each variable and to identify an indicator variable:

```
WORKING-STORAGE SECTION.  
.  
.EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
 01  EMP-ID          PIC S9(8)      USAGE COMP.    ← INTEGER  
 01  EMP-LNAME       PIC X(20).     ← CHARACTER  
 01  SALARY-AMOUNT   PIC S9(6)V(2) USAGE COMP-3. ← DECIMAL  
 01  PROMO-DATE      PIC X(10).    ← DATE  
 01  PROMO-DATE-I    PIC S9(4)      USAGE COMP.    ← Indicator variable  
EXEC SQL END DECLARE SECTION END-EXEC.
```

**Declaring an indicator variable:** An indicator variable must be either a 2 or 4 byte computational (binary) data type. In the example above, PROMO-DATE-I is a valid indicator variable.

**SQLIND data type:** You can declare an indicator variable with the data type SQLIND:

```
05  PROMO_DATE      PIC X(10).      ← DATE  
05  PROMO_DATE_I    SQLIND.        ← Indicator variable
```

The precompiler will substitute PIC S9(8) USAGE COMP in the output source.

The SQLIND data type is primarily for use within bulk structure definitions. In other cases its use is optional.

**Allowable host variable definitions:** A host variable definition may contain:

- PICTURE clause
- USAGE clause

```
DISPLAY  
DISPLAY SIGN LEADING SEPARATE  
COMP  
COMP-11  
COMP-21  
COMP-31  
SQLIND1  
SQLBIN1  
SQLSESS1
```

- VALUE clause<sup>1</sup>
- 88 *condition-name*<sup>1</sup> (any legal COBOL clause)

- OCCURS<sup>1</sup> clause (except within a non-bulk structure)

Within a bulk structure definition, the occurs clause is allowed only on the second-level group element. The following subclauses are also supported but only on the second level group element of a bulk structure:

DEPENDING ON

**Note:** The DEPENDING ON variable is not used in determining the number of rows in the bulk structure.

ASCENDING/DESCENDING KEY

INDEXED BY

- REDEFINES<sup>1</sup> clause (except within a bulk or non-bulk structure)
- BLANK WHEN ZERO<sup>1</sup> (except within a bulk or non-bulk structure)
- SYNCHRONIZED<sup>1</sup> (except within a bulk or non-bulk structure)

(<sup>1</sup> This support is a CA-IDMS extension of ANSI-standard SQL.)

#### 4.3.2.2 Using INCLUDE TABLE

**Output of INCLUDE TABLE:** The CA-IDMS precompiler uses these data type equivalents when directed by an INCLUDE TABLE statement to create a host variable declaration.

CA-IDMS data type	COBOL data type on INCLUDE TABLE
BINARY( <i>n</i> )	PIC X( <i>n</i> )
CHARACTER( <i>n</i> )	PIC X( <i>n</i> )
VARCHAR( <i>n</i> )	-LEN PIC S9(4) COMP -TEXT PIC X( <u>n</u> )
GRAPHIC( <i>n</i> )	PIC G( <i>n</i> ) DISPLAY-1
VARGRAPHIC( <i>n</i> )	-LEN PIC S9(4) COMP -TEXT PIC G( <u>n</u> ) DISPLAY-1
DECIMAL( <i>p,s</i> )	PIC S9( <i>p-s</i> )V9( <i>s</i> ) COMP-3
UNSIGNED DECIMAL( <i>p,s</i> )	PIC 9( <i>p-s</i> )V9( <i>s</i> ) COMP-3
NUMERIC( <i>p,s</i> )	PIC S9( <i>p-s</i> )V9( <i>s</i> ) DISPLAY
UNSIGNED NUMERIC( <i>p,s</i> )	PIC 9( <i>p-s</i> )V9( <i>s</i> ) DISPLAY
DOUBLE PRECISION	COMP-2
FLOAT( <u>n</u> ) where <u>n</u> <= 24 <u>n</u> > 24	COMP-1 COMP-2
REAL	COMP-1
DATE	PIC X(10)
TIME	PIC X(8)
TIMESTAMP	PIC X(26)
SMALLINT	PIC S9(4) COMP
INTEGER	PIC S9(8) COMP
LONGINT	PIC S9(18) COMP
SQLIND	COMP PIC S9(8)

**Default structure:** The default structure created by the INCLUDE statement has these features:

- An 01-level element for the table
- A subordinate element named for each table column, defined with the equivalent program language data type
- An additional element, with the suffix '-I', for each column that allows null values, to be available as an indicator variable
- All element names generated with hyphens to replace underscores that appear in column names, to conform to COBOL naming standards

If you specify a table without a schema name qualifier, you must supply a schema name with a precompiler option. See 5.2.2, “Precompiler options” on page 5-5, for information about precompiler options.

### 4.3.2.3 Defining bulk structures

**About bulk structures:** A bulk structure is a group element or a record which contains a subordinate array for holding multiple occurrences of input or output values. Bulk structures are used in bulk SELECT, INSERT, and FETCH statements for retrieving or storing multiple rows of data.

**Format of a bulk structure:** A bulk structure consists of three levels:

- The highest level is the structure itself (level 01 through 47).
- The second level is a multiply occurring group item (level 02 through 48).
- The third level consists of elementary or variable length data items (variable length data items are group elements consisting of a halfword length field followed by a character or graphics field).

The number, type and order of data items at the lowest level must correspond to the number, data type, and order of column values being retrieved or inserted.

**Bulk structure example:** The following is an example of a valid bulk structure:

```
EXEC SQL BEGIN DECLARE SECTION    END-EXEC.  
02 BULK-DATA.  
04 BULK-ROW OCCURS 20 TIMES.  
05 EMP-ID    PIC 999.  
05 EMP-NAME  PIC X(30).  
05 DEPT-NAME PIC X(30).  
EXEC SQL END DECLARE SECTION    END-EXEC.
```

**Referring to a bulk structure:** When referring to a bulk structure in a SELECT, FETCH, or INSERT statement, the name of the highest level is used:

```
EXEC SQL  
  FETCH EMPCURS BULK :BULK-DATA  
END-EXEC.
```

**Indicator variables:** An indicator variable can be associated with a data item within the structure as follows:

- The indicator variable must immediately follow the data item with which it is associated
- The picture of the indicator variable must be S9(n) where n is between 4 and 8
- The usage of the indicator variable must be SQLIND

On encountering the SQLIND usage, the precompiler interprets the variable as an indicator associated with the preceding variable. SQLIND is replaced with COMP in the generated source.

**Restrictions:** The following COBOL clauses must not appear within a bulk structure definition:

BLANK WHEN ZERO  
JUSTIFIED  
OCCURS (except at the second level)  
REDEFINES  
SIGN  
SYNCHRONIZED

Fillers may appear within the structure; however, their data content is not preserved across a bulk SELECT or FETCH.

**Using INCLUDE TABLE:** A bulk structure can be defined for a given table by using the INCLUDE TABLE statement with a NUMBER OF ROWS clause. The statement in this example will generate a bulk structure capable of holding 20 entries:

```
EXEC SQL
  INCLUDE TABLE EMPLOYEE NUMBER OF ROWS 20
END-EXEC.
```

#### 4.3.2.4 Non-bulk structures and indicator arrays

**About non-bulk structures:** A non-bulk structure is a group element or record which is used to represent a list of host variables within an SQL statement. When reference is made to a non-bulk structure, it is interpreted as a reference to all of the subordinate elements within the structure.

**About indicator arrays:** An indicator array is a group element or record which contains one multiply occurring subordinate element used as an array of indicator variables. Indicator arrays hold indicator values for items within a non-bulk structure.

**Format of a non-bulk structure:** A non-bulk structure consists of two levels:

- The highest level is the structure itself (level 01 through 48)
- The second level consists of elementary or variable length data items (variable length data elements are group elements which consist of a halfword length field followed by a character or graphics field)

The number, type, and order of data items at the lowest level must correspond to the number, data type, and order of column values being retrieved or inserted.

**Non-bulk structure example:** This is an example of a valid non-bulk structure:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  EMP-INFO.
    05  EMP-ID    PIC 999.
    05  EMP-NAME  PIC X(30).
    05  DEPT-NAME PIC X(30).
EXEC SQL END DECLARE SECTION END-EXEC.
```

**Format of an indicator array:** An indicator array consists of two levels:

- The highest level represents the entire array (level 01 through 48)
- The second level is a multiply occurring element that defines a halfword or fullword field

This is an example of a valid indicator array:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
      02  INDS.  
          04  IND SQLIND OCCURS 20 TIMES.  
EXEC SQL END DECLARE SECTION END-EXEC.
```

**Referring to a non-bulk structure:** A non-bulk structure can be referred to anywhere a list of host variables can be specified:

- The INTO clause of a SELECT or FETCH statement; for example:

```
EXEC SQL  
      FETCH EMPCURS INTO :EMP-INFO  
END-EXEC.
```

- The VALUES clause of an INSERT statement

Unlike bulk processing, a single SQL statement can contain more than one reference to a non-bulk structure. Each such reference is interpreted as a list of host variable references. The union of all such host variables together with any elementary host variables must correspond to a single result row (or input row, in the case of an INSERT statement).

**Referring to an indicator array:** To associate indicator variables with the elements of the non-bulk structure, the name of an indicator array is specified immediately following the name of the non-bulk structure:

```
EXEC SQL  
      FETCH EMPCURS INTO :INFO :INDS  
END-EXEC.
```

**Note:** Either the name of the group or its subordinate element may be used to refer to an indicator array.

**Association of indicator variables and non-bulk structure elements:** The number of occurrences in the indicator array need not be the same as the number of elements in the non-bulk structure with which it is used. If there are more indicators than elements, the remaining indicators are ignored, although their contents are not necessarily preserved. If there are fewer indicators than elements, an indicator is associated with each element in the structure until all indicators are assigned. The remaining elements do not have associated indicators. This may result in an error if an attempt is made to return a null value into an element with no associated indicator.

**Restrictions:** The following COBOL clauses must not appear within a non-bulk structure definition:

- BLANK WHEN ZERO
- JUSTIFIED

- OCCURS
- REDEFINES
- SIGN
- SYNCHRONIZED

Fillers having a character data type may appear within the structure. However, their data content is not preserved across a SELECT or FETCH.

**Note:** Unless the included table has no nullable columns an INCLUDE TABLE table-name precompiler directive cannot be used to define the non-bulk structure; any nullable column would cause the precompiler to insert an associated indicator variable which makes the structure unusable for reference in the FETCH statement.

### 4.3.3 Referring to host variables

**What you can do:** CA-IDMS supports references to host variables in SQL statements. The host variable name must be prefixed with a colon (:).

►►See 2.2.3, “Referring to host variables” on page 2-7, for more information.

CA-IDMS also supports references to:

- Subordinate elements which may require qualification for uniqueness
- Subscripted elements

**Qualifying host variable names:** CA-IDMS supports two methods of qualifying host variable names.

For example, assume these host variable definitions:

```
01 EMP
    03 HIRE-DATE
    .
    .
01 MGR
    03 HIRE-DATE
    .
    .
```

The method of qualifying HIRE-DATE in either of the following examples is valid:

```
EXEC SQL
    SELECT...
        INTO :HIRE-DATE OF EMP
-----
EXEC SQL
    SELECT...
        INTO :EMP.HIRE-DATE
```

**Subscripted variable names:** A CA-IDMS extension of ANSI-standard SQL supports host variable arrays for use in bulk processing. By further extension of ANSI-standard SQL, CA-IDMS supports reference to a subscripted variable in a host variable array.

All of the following are valid host variable references:

```
:DIV-CODE(1)  
:DIV-CODE (15)  
:DIV-CODE(SUB1)  
:DIV-CODE(SUB1,SUB2)
```

#### 4.3.4 Including SQL Communication Areas

**Declaring SQL Communication Areas:** CA-IDMS provides these ways of including the SQL Communication Areas in a COBOL program:

- The program can declare the host variable SQLSTATE in the WORKING-STORAGE SECTION:

```
01 SQLSTATE      PIC X(5).
```

**Note:** SQLSTATE does not have to be defined inside an SQL declare section.

- The program can declare the host variable SQLCODE in the WORKING-STORAGE SECTION:

```
01 SQLCODE      PIC S9(8)  
          USAGE COMP.
```

**Note:** SQLCODE does not have to be defined inside an SQL declare section.

- The precompiler automatically includes the communication areas at the end of the WORKING-STORAGE section in any program that contains embedded SQL statements

- The program can issue this precompiler directive:

```
EXEC SQL  
  INCLUDE SQLCA  
END-EXEC.
```

Using the INCLUDE statement to declare the SQLCA is a CA-IDMS extension of ANSI-standard SQL.

**SQLCA structure:** This is the COBOL format of the SQLCA:

```

01 SQLCA.
  02 SQLCAID          PIC X(8).
  02 SQLCODE          PIC S9(8) COMP.
  02 SQLCSID          PIC X(8).

  02 SQLCINFO.
    03 SQLCERC          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCNRP          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCSER          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 SQLCLNO          PIC S9(8) COMP.
    03 SQLCMCT          PIC S9(8) COMP.
    03 SQLCARC          PIC S9(8) COMP.
    03 SQLCFJB          PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.
    03 FILLER           PIC S9(8) COMP.

  02 SQLCINF2 REDEFINES SQLCINFO.
    03 SQLERRD          PIC S9(8) COMP.
                           OCCURS 12.

  02 SQLCMMSG.
    03 SQLCERL          PIC S9(8) COMP.
    03 SQLERM           PIC X(256).

  02 SQLCMMSG2 REDEFINES SQLCMMSG.
    03 FILLER           PIC X(2).

    03 SQLERRM.
      04 SQLCERRML        PIC S9(4) COMP.
      04 SQLERRMC         PIC X(256).

  02 SQLSTATE          PIC X(5).

  02 FILLER            PIC X(11).

  02 SQLWORK            PIC X(16).

  02 SQLCWRK2 REDEFINES SQLWORK.
    03 SQLERRP.
      04 SQLCVAL          PIC X(5).
      04 FILLER           PIC X(3).

    03 SQLWARN.
      04 SQLWARN0          PIC X(1).
      04 SQLWARN1          PIC X(1).
      04 SQLWARN2          PIC X(1).
      04 SQLWARN3          PIC X(1).
      04 SQLWARN4          PIC X(1).
      04 SQLWARN5          PIC X(1).
      04 SQLWARN6          PIC X(1).
      04 SQLWARN7          PIC X(1).  


```

Included by the precompiler for DB2 compatibility; not used by CA-IDMS

### 4.3.5 Copying information from the dictionary

You can use these precompiler directives to instruct the precompiler to copy entities from the dictionary into the COBOL application program:

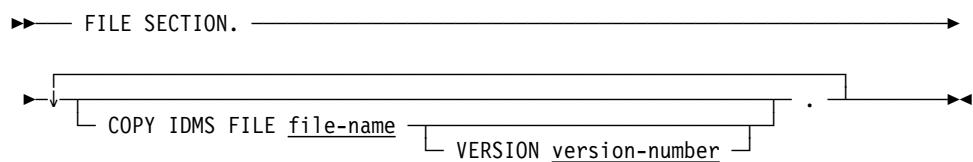
- COPY IDMS FILE
- COPY IDMS RECORD
- COPY IDMS MODULE
- INCLUDE *module-name*

## 4.3.6 COPY IDMS FILE statement

### 4.3.6.1 Purpose

COPY IDMS FILE statements copy file descriptions from the dictionary into the program. Each COPY IDMS FILE statement generates the file definition that includes record size, block size, and recording mode from the dictionary. Also, any records included in the file through the Integrated Data Dictionary (IDD) facilities are also copied.

### 4.3.6.2 Syntax



### 4.3.6.3 Parameters

#### file-name

Copies the description of a non-CA-IDMS file into the DATA DIVISION.

*File-name* is either the primary name or a synonym for a file defined in the dictionary.

#### VERSION version-number

Qualifies *file-name* with a version number. *Version-number* must be an integer in the range 1 through 9999 and defaults to the highest version number defined in the dictionary for *file-name*.

### 4.3.6.4 Usage

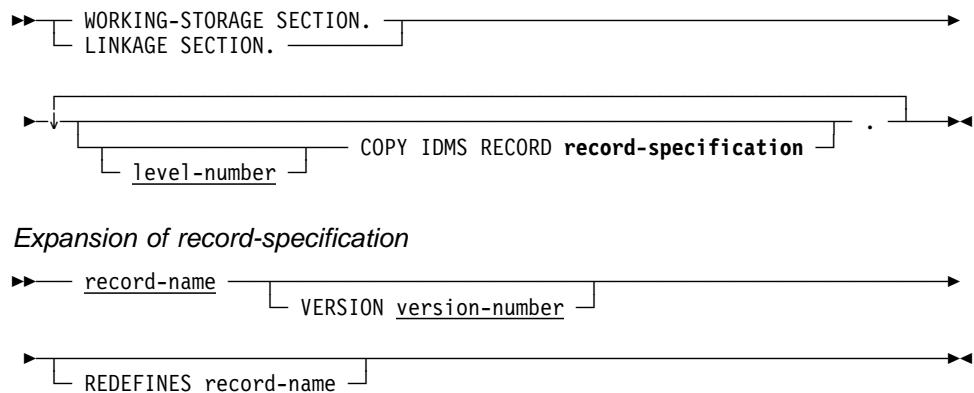
The FILE SECTION of the DATA DIVISION can include one or more COPY IDMS FILE statements.

## 4.3.7 COPY IDMS RECORD statement

### 4.3.7.1 Purpose

Allows you to copy a record description from the dictionary into the DATA DIVISION of a COBOL program at the location of the COPY IDMS statement.

#### 4.3.7.2 Syntax



#### 4.3.7.3 Parameters

##### level-number

Instructs the precompiler to copy the descriptions into the program at a level other than that originally specified for the description in the dictionary. *Level-number* must be an integer in the range 01 through 48.

If *level-number* is specified, the first level will be copied to the level specified by *level-n*; all other levels will be adjusted accordingly. If *level-n* is not specified, the descriptions copied will have the same level numbers as originally specified in the dictionary.

##### record-name

Specifies the name of the record to be copied. *Record-name* can be either the primary name or a synonym for a record stored in the dictionary.

##### version-number

Qualifies dictionary records with a version number. *Version-number* must be an integer in the range 1 through 999.

If *version-number* is not specified, the record that is copied will be the record synonym for the named record that is the highest version defined for COBOL.

##### REDEFINES record-name

Copies a record description to an area previously defined by another record description. Two record descriptions can thus provide alternative definitions of the same storage location.

#### 4.3.7.4 Usage

*Invalid descriptors:* The program can copy a record definition from the dictionary and use the record elements as host variables in embedded SQL.

If you declare host variables by copying a record description from the dictionary, you must observe all rules regarding host variable declarations.

*Placement:* You can place COPY IDMS RECORD statements in any area of the DATA DIVISION that COBOL allows record definitions.

*VALUE clauses:* If the dictionary record is to be copied into the LINKAGE SECTION and includes VALUE clauses, the VALUE clauses are not copied.

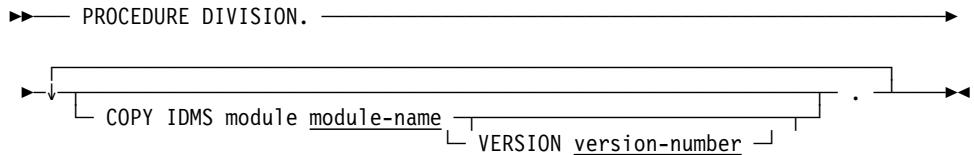
*Using COPY IDMS RECORD for host variables:* If the record to copy contains fields that the program may reference as host variables, you must include the COPY IDMS RECORD statement in an SQL declaration section.

## 4.3.8 COPY IDMS MODULE statement

### 4.3.8.1 Purpose

Copies source statements from a module stored in the data dictionary into the source program.

### 4.3.8.2 Syntax



### 4.3.8.3 Parameters

#### module-name

Specifies the name of a module previously defined in the dictionary.

#### version-number

Qualifies *module-name* with a version number. *Version-number* must be an integer in the range 1 through 9999.

If *version-number* is not specified, the record copied will be the highest version of the named module defined in the dictionary for COBOL.

### 4.3.8.4 Usage

*Placement:* The unmodified module is placed into the program by the precompiler at the location of the request. The location of the request is usually in the PROCEDURE DIVISION, but it can be anywhere that is appropriate for the contents of the module to be included in the program.

*Nesting modules:* COPY IDMS MODULE statements can be nested (that is, a statement invoked by a COPY IDMS MODULE entry can itself be a COPY IDMS MODULE statement). However, you must ensure that a copied module does not, in turn, copy itself.

### 4.3.9 INCLUDE module-name statement

The INCLUDE *module-name* statement is equivalent to a COPY IDMS MODULE statement in which the version number is omitted.

- For complete documentation of this statement, refer to *CA-IDMS SQL Reference Guide*.

### 4.3.10 Non-SQL precompiler directives

The CA-IDMS precompiler accepts several directives that are not associated with SQL statements and host variable declarations. These include:

- RETRIEVAL — Specifies that the precompiler should ready the area of the dictionary containing data definitions in retrieval mode, allowing concurrent update of the area by other transactions
  - PROTECTED — Specifies that the precompiler should ready the area of the dictionary containing data definitions in update mode, preventing concurrent update of the area by other transactions
  - NO-ACTIVITY-LOG — Suppresses the logging of program activity statistics
  - DMLIST/NODMLIST — Specifies generation or no generation of a source listing for the statements that follow
- For complete documentation of non-SQL precompiler directives, see Appendix C, “Precompiler Directives.”

## 4.4 Using SQL in a PL/I application program

This section presents information that is specific to embedding SQL in a PL/I application program.

**Note:** Refer to *CA-IDMS DML Reference - PL/I* for documentation of all aspects of PL/I application programming in the CA-IDMS environment.

### 4.4.1 Embedding SQL statements

**Requirements:** To embed an SQL statement in a PL/I program, you must:

- Include an SQLXQ1 declaration
- Observe PL/I margin requirements (columns 2 to 72)
- Use statement delimiters

**Options:** You can use SQL conventions to:

- Continue an SQL statement on the next line
- Insert comments in an SQL statement

You can use a precompiler-directive statement to copy SQL statements in a module from the dictionary into the program.

#### 4.4.1.1 Declaring SQLXQ1

PL/I applications with embedded SQL must include the SQLXQ1 ENTRY statement. The syntax for this statement is:

```
► ┌─ DECLARE ─┐ SQLXQ1 ENTRY OPTIONS (INTER, ASSEMBLER); ──────────────────►
   DCL
```

#### 4.4.1.2 Delimited, continued, and commented statements

**Using SQL statement delimiters:** When you embed an SQL statement in a PL/I application program, you must use these statement delimiters:

- Begin each SQL statement with **EXEC SQL**
- End each SQL statement with ;

An EXEC SQL delimiter must be preceded by either a PL/I label or the ; character.

The example below shows the use of SQL statement delimiters:

```
EXEC SQL INSERT INTO DIVISION VALUES ('D07','LEGAL',1234) ;
```

The statement text can be on the same line as the delimiters.

**Continuing statements:** You can write SQL statements on one or more lines. No special character is required to show that a statement continues on the next line if you split the statement before or after any keyword, value, or delimiter.

**Inserting SQL comments:** To include comments within SQL statements embedded in a PL/I program, you can:

- Use the PL/I comment delimiters /\* and \*/
- Use the SQL comment characters, two consecutive hyphens (--), on an SQL statement line following the statement text

A comment that begins with the SQL comment characters (--) terminates at the end of the line (column 72).

You cannot use SQL comment characters to insert a comment in the middle of a string constant or delimited identifier.

The example below shows both methods of inserting comments within an embedded SQL statement:

```
EXEC SQL
/***** PERFORM UPDATE ON ACTIVE EMPLOYEES ONLY *****/
    UPDATE BENEFITS
        SET VAC_ACCRUED = VAC_ACCRUED + 10, -- Add 10 hours vacation
            SICK_ACCRUED = SICK_ACCRUED + 1 -- Add 1 sick day
        WHERE EMP_ID IN
            (SELECT EMP_ID FROM EMPLOYEE
             WHERE STATUS = 'A') ;
```

## 4.4.2 Defining host variables

**What you declare:** Within an SQL declare section, you specify the name, level, and data type of host variables using standard PL/I data declarative statements and observing these guidelines:

- A host variable **name** must conform to PL/I rules for forming variable names
- The **level** number is in the range of 1 to 255
- The **data type** of the host variable

### 4.4.2.1 Using PL/I declarations

**Equivalent column data types:** This table shows data types of PL/I host variables that are valid in an SQL declare section and equivalent to CA-IDMS table column data types:

Equivalent PL/I data type	CA-IDMS data type
CHAR ( <i>n</i> )	CHAR( <i>n</i> )
CHAR ( <i>n</i> ) VAR	VARCHAR( <i>n</i> )
FIXED DECIMAL ( <i>p,s</i> )	DECIMAL( <i>p,s</i> )
FLOAT BINARY ( <u>n</u> ) where <u>n</u> <= 24 where <u>n</u> > 24	REAL DOUBLE PRECISION
FLOAT DECIMAL ( <u>n</u> ) where <u>n</u> <= 6 where <u>n</u> > 6	REAL DOUBLE PRECISION
FIXED BINARY (15)	SMALLINT
FIXED BINARY (31)	INTEGER
CHAR ( <i>n</i> )	BINARY( <i>n</i> ) <sup>1</sup>
GRAPHIC ( <i>n</i> )	GRAPHIC( <i>n</i> ) <sup>1</sup>
GRAPHIC ( <i>n</i> ) VAR	VARGRAPHIC( <i>n</i> ) <sup>1</sup>
CHAR (10)	DATE <sup>1</sup>
CHAR (8)	TIME <sup>1</sup>
CHAR (26)	TIMESTAMP <sup>1</sup>
SQLBIN ( <i>n</i> )	BINARY( <i>n</i> )

**Note:**

<sup>1</sup>This data type is a CA-IDMS extension of ANSI-standard SQL.

► Refer to *CA-IDMS SQL Reference Guide* for documentation of CA-IDMS data types.

**Data types not supported:** The table below shows CA-IDMS data types for which there are no equivalent data types in PL/I that are valid in an SQL declare section. The table shows compatible PL/I data types that are valid in host variable declarations; however, accessing a column that has no equivalent data type may result in an error if a data value is not convertible between the two data types.

Compatible PL/I data type	CA-IDMS data type
FIXED BINARY (31)	LONGINT
FIXED DECIMAL ( <i>p,s</i> )	NUMERIC( <i>p,s</i> )
FIXED DECIMAL ( <i>p,s</i> )	UNSIGNED NUMERIC( <i>p,s</i> )
FIXED DECIMAL ( <i>p,s</i> )	UNSIGNED DECIMAL( <i>p,s</i> )

**Host variable declaration example:** In this example, the SQL declare section defines host variables, including one indicator variable, using standard PL/I data declarations. The example is annotated to show the equivalent column data type for each variable and to identify an indicator variable:

```
WORKING-STORAGE SECTION.  
.  
.EXEC SQL BEGIN DECLARE SECTION ;  
DECLARE 1  EMP_ID          FIXED BINARY (31) ;      ← INTEGER  
DECLARE 1  EMP_LNAME        CHAR (20) ;             ← CHARACTER  
DECLARE 1  SALARY_AMOUNT    FIXED DECIMAL (6,2) ;   ← DECIMAL  
DECLARE 1  PROMO_DATE       CHAR (10) ;              ← DATE  
DECLARE 1  PROMO_DATE_I     FIXED BINARY (31) ;      ← Indicator variable  
EXEC SQL END DECLARE SECTION ;
```

**Declaring an indicator variable:** An indicator variable must be either FIXED BINARY (15) or FIXED BINARY (31) data type. In the example above, PROMO\_DATE\_I is an indicator variable for PROMO\_DATE.

**SQLIND data type:** You can declare an indicator variable with the data type SQLIND:

```
DECLARE 1  PROMO_DATE       CHAR (10) ;              ← DATE  
DECLARE 1  PROMO_DATE_I     SQLIND ;                ← Indicator variable
```

The precompiler will substitute a FIXED BINARY (31) in the output source.

**Note:** The SQLIND data type is primarily for use within bulk structure definitions. In other cases its use is optional.

**Allowable host variable definitions:** A host variable definition must contain a data type declaration and may contain an occurrence count. No other declarations are supported.

#### 4.4.2.2 Using INCLUDE TABLE

**Output of INCLUDE TABLE:** The CA-IDMS precompiler uses these data type equivalents when directed by an INCLUDE TABLE statement to create a host variable declaration.

CA-IDMS data type	PL/I data type on INCLUDE TABLE
BINARY( <i>n</i> )	CHAR ( <i>n</i> )
CHARACTER( <i>n</i> )	CHAR ( <i>n</i> )
VARCHAR( <i>n</i> )	CHAR ( <i>n</i> ) VAR
GRAPHIC( <i>n</i> )	GRAPHIC ( <i>n</i> )
VARGRAPHIC( <i>n</i> )	GRAPHIC ( <i>n</i> ) VAR
DECIMAL( <i>p,s</i> )	FIXED DECIMAL ( <i>p,s</i> )
UNSIGNED DECIMAL( <i>p,s</i> )	FIXED DECIMAL ( <i>p,s</i> )
NUMERIC( <i>p,s</i> ) <sup>1</sup>	FIXED DECIMAL ( <i>p,s</i> )
UNSIGNED NUMERIC( <i>p,s</i> )	FIXED DECIMAL ( <i>p,s</i> )
DOUBLE PRECISION	FLOAT BINARY (53)
FLOAT ( <i>n</i> ) where <i>n</i> <= 24 where <i>n</i> > 24	FLOAT BINARY (21) FLOAT BINARY (53)
REAL	FLOAT BINARY (21)
DATE	CHAR (10)
TIME	CHAR (8)
TIMESTAMP	CHAR (26)
SMALLINT	FIXED BINARY (15)
INTEGER	FIXED BINARY (31)
LONGINT	FIXED BINARY (31)
SQLIND	FIXED BINARY (31)

**Default structure:** The default structure created by the INCLUDE statement has these features:

- A level 1 element for the table
- A level 2 subordinate element named for each table column, defined with the equivalent program language data type
- An additional level 2 element, with the suffix '\_I', for each column that allows null values, to be available as an indicator variable

If you specify a table without a schema name qualifier, you must supply a schema name with a precompiler option in the JCL.

►► See 5.2.2, “Precompiler options” on page 5-5, for information about precompiler options.

#### 4.4.2.3 Defining bulk structures

**About bulk structures:** A bulk structure is a group element or a record which contains a subordinate array for holding multiple occurrences of input or output values. Bulk structures are used in bulk SELECT, INSERT, and FETCH statements for retrieving or storing multiple rows of data.

**Format of a bulk structure:** A bulk structure consists of three levels:

- The highest level is the structure itself (level 01 through 253)
- The second level is a multiply-occurring group item (level 02 through 254)
- The third level consists of elementary or variable length data items

The number, type and order of data items at the lowest level must correspond to the number, data type, and order of column values being retrieved or inserted.

**Bulk structure example:** The following is an example of a valid bulk structure:

```
EXEC SQL BEGIN DECLARE SECTION;
DCL 1 BULK_DATA,
      4 BULK_ROW (20),
      5 EMP_ID FIXED DECIMAL(3),
      5 EMP_NAME CHAR(30),
      5 DEPT_NAME CHAR(30);
EXEC SQL END DECLARE SECTION;
```

**Referring to a bulk structure:** When referring to a bulk structure in a SELECT, FETCH, or INSERT statement, the name of the highest level is used:

```
EXEC SQL
  FETCH EMPCURS BULK :BULK_DATA;
```

**Indicator variables:** An indicator variable can be associated with a data item within the structure as follows:

- The indicator variable must immediately follow the data item with which it is associated
- The data type of the indicator variable must be SQLIND

On encountering the SQLIND data type, the precompiler interprets the variable as an indicator associated with the preceding variable. SQLIND is replaced with BINARY FIXED(31) in the generated source.

**Restrictions:** A subscripted data element may not appear within the lowest level of a bulk structure.

**Using INCLUDE TABLE:** A bulk structure can be defined for a given table by using the INCLUDE TABLE statement with a NUMBER OF ROWS clause. The statement in this example will generate a bulk structure capable of holding 20 entries:

```
EXEC SQL
  INCLUDE TABLE EMPLOYEE NUMBER OF ROWS 20;
```

### 4.4.3 Referring to host variables

**What you can do:** CA-IDMS supports references to host variables in SQL statements. The host variable name must be prefixed with a colon (:).

►► See 2.2.3, “Referring to host variables” on page 2-7, for more information.

CA-IDMS also supports references to:

- Subordinate elements which may require qualification for uniqueness
- Subscripted elements

**Qualifying host variable names:** You can use the group name to qualify the element name of a host variable.

For example, assume these host variable definitions:

```
DECLARE 1 EMP,  
        2 HIRE_DATE  
.  
.  
DECLARE 1 MGR,  
        2 HIRE_DATE  
.  
.  
.
```

You can qualify HIRE\_DATE as in this example:

```
EXEC SQL  
  SELECT...  
    INTO :EMP.HIRE_DATE ;
```

**Subscripted variable names:** A CA-IDMS extension of ANSI-standard SQL supports host variable arrays for use in bulk processing. By further extension of ANSI-standard SQL, CA-IDMS supports reference to a subscripted variable in a host variable array.

All of the following are valid host variable references:

```
:DIV-CODE(1)  
:DIV-CODE (15)  
:DIV-CODE(SUB1)  
:DIV-CODE(SUB1,SUB2)
```

#### 4.4.4 Including SQL Communication Areas

**Declaring SQL Communication Areas:** CA-IDMS provides these ways of including the SQL Communication Areas in a PL/I program:

- The program can declare the host variable SQLSTATE:

```
EXEC SQL BEGIN DECLARE SECTION ;
DECLARE   SQLSTATE      CHARACTER(5) ;
EXEC SQL END DECLARE SECTION ;
```

- The program can declare the host variable SQLCODE:

```
EXEC SQL BEGIN DECLARE SECTION ;
DECLARE   SQLCODE      FIXED BINARY (31) ;
EXEC SQL END DECLARE SECTION ;
```

- The program can issue this directive:

```
EXEC SQL INCLUDE SQLCA ;
```

Using the INCLUDE statement to declare the SQLCA is a CA-IDMS extension of ANSI-standard SQL.

**SQLCA structure:** This is the PL/I format of the SQLCA:

```

DECLARE 1 SQLCA,
        2 SQLCAID      CHARACTER (8),
        2 SQLCODE      FIXED BINARY (31),
        2 SQLCSID      CHARACTER (8),
        2 SQLCINFO,
          3 SQLCERC      FIXED BINARY (31),
          3 FILLERnnnnn  FIXED BINARY (31),
          3 SQLCNRP      FIXED BINARY (31),
          3 FILLERnnnnn  FIXED BINARY (31),
          3 SQLCSER      FIXED BINARY (31),
          3 FILLERnnnnn  FIXED BINARY (31),
          3 SQLCLNO      FIXED BINARY (31),
          3 SQLCMCT      FIXED BINARY (31),
          3 SQLCARC      FIXED BINARY (31),
          3 SQLCFJB      FIXED BINARY (31),
          3 FILLERnnnnn  FIXED BINARY (31),
          3 FILLERnnnnn  FIXED BINARY (31),
        2 SQLCMMSG,
          3 SQLCERL      FIXED BINARY (31),
          3 SQLCERM      CHARACTER (256),
        2 SQLSTATE      CHARACTER (5),
        2 FILLERnnnnn  CHARACTER (11),
        2 SQLWORK       CHARACTER (16) ;

DECLARE 1 SQLCINF2 BASED (ADDR(SQLCINFO)),
        2 SQLERRD      FIXED BINARY (31),

DECLARE 1 SQLCMMSG2 BASED(ADDR(SQLCMMSG)),
        2 FILLERnnnnn  CHARACTER (2),
        2 SQLERRM,
          3 SQLERRML     CHARACTER (5),
          3 SQLERRMC     CHARACTER (256) ;

DECLARE 1 SQLCWRK2 BASED(ADDR(SQLWORK)),
        2 SQLERRP,
          3 SQLCVAL      CHARACTER (5),
          3 FILLERnnnnn  CHARACTER (3),
        2 SQLWARN,
          3 SQLWARN0     CHARACTER (1),
          3 SQLWARN1     CHARACTER (1),
          3 SQLWARN2     CHARACTER (1),
          3 SQLWARN3     CHARACTER (1),
          3 SQLWARN4     CHARACTER (1),
          3 SQLWARN5     CHARACTER (1),
          3 SQLWARN6     CHARACTER (1),
          3 SQLWARN7     CHARACTER (1) ;

```

Included by the precompiler for DB2 compatibility; not used by CA-IDMS

#### 4.4.5 Including information from the dictionary

You can use these precompiler directive statements to instruct the precompiler to copy entities from the dictionary into the PL/I application program:

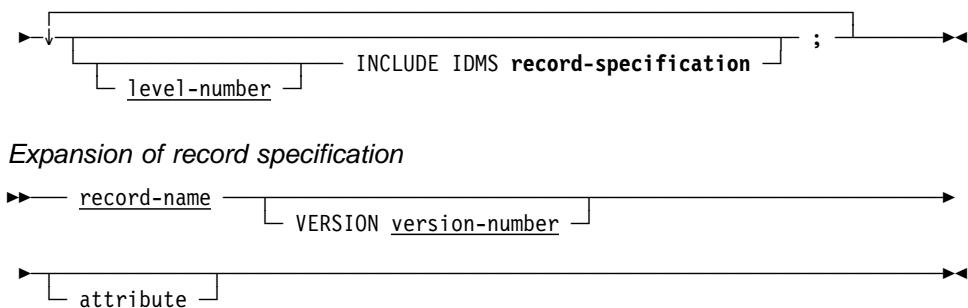
- INCLUDE IDMS *record-name*
- INCLUDE IDMS MODULE *module-name*
- INCLUDE *module-name*

## 4.4.6 INCLUDE IDMS record statement

### 4.4.6.1 Purpose

You can code INCLUDE IDMS statements in your application program to copy record descriptions into the program.

### 4.4.6.2 Syntax



### 4.4.6.3 Parameters

#### level-number INCLUDE IDMS

Instructs the precompiler to copy one or more record descriptions into your program at the location of the INCLUDE IDMS statement.

The optional *level-number* clause instructs the precompiler to copy descriptions into your program at a different level than the level specified in the data dictionary. *Level-number* must be an integer in the range 01 through 99. If your program specifies *level-number*, the DML precompiler copies the first level of code to the level specified by *level-number* and adjusts all other levels accordingly. If your program does not specify *level-number*, the descriptions copied by the DML precompiler have the same level numbers as originally specified in the dictionary.

#### record-name

Specifies the name of the record to be copied. It can be the primary name of a record stored in the data dictionary, or a synonym.

#### VERSION version-number

Optionally qualifies IDD records with a version number. *Version-number* must be an integer in the range 1 through 9999. *Version-number* defaults to the highest version number of the record defined in the data dictionary for the language and operating mode under which the program compiles.

#### attribute

Optionally allows you to instruct the DML precompiler to include PL/I attributes in the PL/I DECLARE statement. The DML precompiler generates the PL/I DECLARE statement for the record that you specify in *record-name*.

#### 4.4.6.4 Usage

*Using included records as host variables:* The program can copy a record definition from the dictionary and use the record elements as host variables in embedded SQL.

If you declare host variables by copying a record description from the dictionary, the following descriptors should not appear in the record definition:

```
REDEFINES
SYNC
```

### 4.4.7 INCLUDE IDMS MODULE statement

#### 4.4.7.1 Purpose

INCLUDE IDMS (*module-name*) copies procedure source statements defined by the database administrator as modules in the dictionary.

#### 4.4.7.2 Syntax



#### 4.4.7.3 Parameters

##### INCLUDE IDMS (module-name)

Copies procedure source statements defined by the DBA as modules in the dictionary. *Module-name* specifies the name of a module previously defined using the DDDL compiler.

► Refer to *CA-IDMS IDD DDDL Reference Guide* for information about the DDDL compiler.

The available PL/I standard modules are:

- IDMS\_STATUS
- IDMS\_STATUS (mode is IDMS\_DC)

The DML precompiler inserts the module into your program at the location of the INCLUDE IDMS MODULE statement, without modification.

You can nest INCLUDE IDMS MODULE statements. This means that code invoked by an INCLUDE IDMS MODULE entry can itself contain INCLUDE IDMS MODULE statements. However, make sure that a copied module does not copy itself.

#### **VERSION *version-number***

Optionally qualifies *module-name* with a version number. *Version-number* must be an integer in the range 1 through 9999.

There are two defaults for *version-number*, depending on whether:

- There is a version of the module that you name with *module-name* which is operating-mode-specific. In this case, the default is the version number of this module. If there are two or more mode-specific versions of the module, *version-number* defaults to the highest version number among these versions.
- There is a version of the module that you name with *module-name* which is non-operating-mode-specific, and there exists no operating-mode-specific version. In this case, the default is the version number of this module. If there are two or more non-mode-specific versions of the module, *version-number* defaults to the highest version number among these versions.

If no version of the module exists in the dictionary, an error condition results.

#### **4.4.8 INCLUDE *module-name* statement**

The INCLUDE *module-name* statement is equivalent to an INCLUDE IDMS MODULE statement in which the version number is omitted.

►► Refer to *CA-IDMS SQL Reference Guide* for complete documentation of this statement.

#### **4.4.9 Non-SQL precompiler directives**

The CA-IDMS precompiler accepts several directives that are not associated with SQL statements and host variable declarations. These include:

- RETRIEVAL — Specifies that the precompiler should ready the area of the dictionary containing data definitions in retrieval mode, allowing concurrent update of the area by other transactions
- PROTECTED — Specifies that the precompiler should ready the area of the dictionary containing data definitions in update mode, preventing concurrent update of the area by other transactions
- NO-ACTIVITY-LOG — Suppresses the logging of program activity statistics
- DMLIST/NODMLIST — Specifies generation or no generation of a source listing for the statements that follow

►► See Appendix C, “Precompiler Directives,” for complete documentation of non-SQL precompiler directives.

# Chapter 5. Preparing and Executing the Program

---

5.1 About this chapter . . . . .	5-3
5.2 Precompiling the program . . . . .	5-4
5.2.1 About the precompiler . . . . .	5-4
5.2.2 Precompiler options . . . . .	5-5
5.2.2.1 Syntax . . . . .	5-5
5.2.2.2 Parameters . . . . .	5-6
5.3 Compiling the program . . . . .	5-10
5.4 Creating the access module . . . . .	5-11
5.4.1 Overriding access module defaults . . . . .	5-11
5.4.2 Altering an access module . . . . .	5-15
5.5 Executing the application . . . . .	5-16
5.6 Testing the access module . . . . .	5-17
5.7 Debugging the application . . . . .	5-18
5.7.1 The Command Facility . . . . .	5-18
5.7.2 SQL trace facility . . . . .	5-19
5.7.3 EXPLAIN statement . . . . .	5-19
5.7.4 Online debugger . . . . .	5-20



## 5.1 About this chapter

To put your source program into executable form, take the following steps:

1. Precompile the program
2. Compile and link edit the program
3. Create the access module
4. Execute and debug the program

If you are using CA-ADS, the CA-ADS compiler ADSC performs steps 1 and 2.

## 5.2 Precompiling the program

You precompile the program to separate SQL statements from the rest of the program and to replace the SQL statements in the source program module with calls to the DBMS.

### 5.2.1 About the precompiler

**Why you precompile:** SQL is a database sublanguage that is not known to the language compiler. The CA-IDMS precompiler:

- Checks the syntax of embedded SQL statements
- Modifies the source code by:
  - Replacing SQL statements in the source program with program language calls to the DBMS
  - Executing precompiler directives
- Stores a relational command module (RCM) for the program if no errors occur in precompiling

**When to precompile:** Once you have precompiled a program, you must precompile it again after any changes to either host language or embedded SQL statements. When you precompile a program that was previously precompiled, the DBMS rebuilds the RCM only if one or more SQL statements in the program have changed.

**Note:** After a program has been precompiled, you can make global changes to the schema-name qualifiers of tables and views in embedded SQL statements when you create the access module. If instead you modify the SQL statements in the source program, you must precompile the program again.

For more information, see the documentation of schema-name mapping for tables and views in 5.4, “Creating the access module” on page 5-11.

**How you precompile:** You precompile the program by submitting a batch job.

►► See Appendix A, “Sample JCL” for precompiler JCL.

You can specify parameters in the precompiler JCL that determine how the precompiler executes.

►► See 5.2.2, “Precompiler options” on page 5-5, for documentation of precompiler parameters.

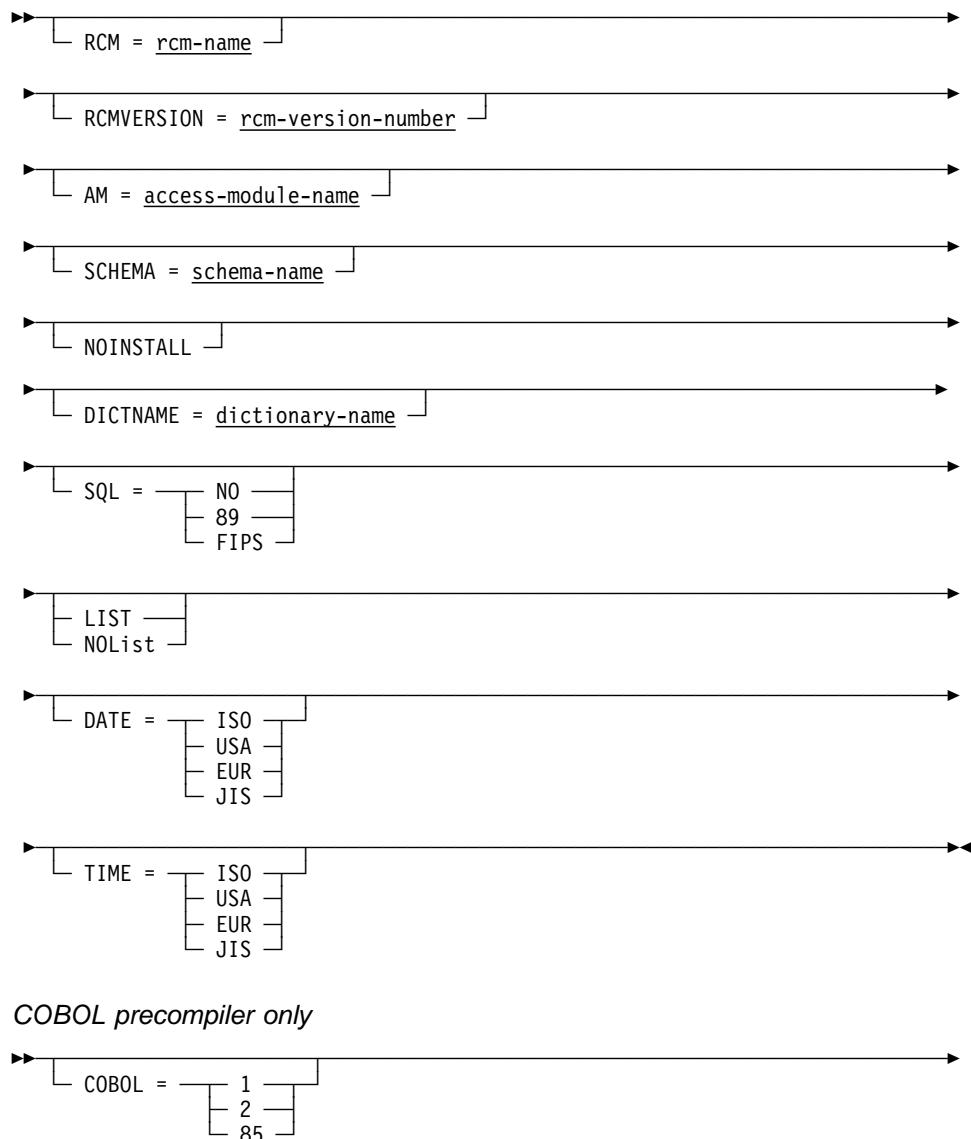
**Authorization:** To execute the precompiler, you must have:

- The authority to precompile the program if program registration is in effect for the dictionary
- User authority to precompile against the dictionary
- SELECT privilege on tables named in INCLUDE TABLE statements

## 5.2.2 Precompiler options

### 5.2.2.1 Syntax

This is the expansion of *precompiler-options* in the precompiler EXEC PGM statement in JCL. These are not positional parameters:



### 5.2.2.2 Parameters

#### **RCM = rcm-name**

Specifies the name of the RCM created for the program by the precompiler.

This parameter must be specified for all host language programs except COBOL.

If this RCM is not specified to the COBOL precompiler, the RCM name is the program name identified in the program source. If the name is not identified in the program, you must specify an RCM parameter.

#### **RCMVERSION = rcm-version-number**

Specifies the version number of the RCM created for the program by the precompiler.

If RCMVERSION is not specified, the version number defaults to 1. If an RCM with the same version number already exists in the dictionary, the precompiler replaces the existing RCM.

#### **AM = access-module-name**

Specifies the name of the access module to be executed for the program at runtime.

The program can override this specification at runtime by issuing a SET ACCESS MODULE statement.

If this parameter is not specified, the access module name defaults to *rcm-name*.

The access module specified in *access-module-name* does need not exist when the program is precompiled. However, if the access module does not exist when the program is executed, an *invalid SQL statement identifier* error occurs.

#### **SCHEMA = schema-name**

Specifies the default schema-name qualifier for the precompiler to use when processing an INCLUDE TABLE statement that does not supply a qualifier.

If an INCLUDE TABLE statement supplies a qualifier, the SCHEMA parameter is ignored for that table.

If SCHEMA is not specified and an INCLUDE TABLE statement does not supply a qualifier, the precompiler returns an error.

#### **NOINSTALL**

Specifies that the precompiler should only check syntax.

If this parameter is specified, the precompiler does not store the RCM.

If this parameter is not specified and the precompiler executes without errors, the precompiler stores the RCM.

#### **DICTNAME = dictionary-name**

Specifies the name of the dictionary the precompiler should access.

If this parameter is not specified, the precompiler defaults to the dictionary specified in the DICTNAME parameter of the SYSIDMS statement in the precompiler JCL.

►► See Appendix A, “Sample JCL,” for sample precompiler JCL.

If this parameter is not specified and there is no SYSIDMS DICTNAME parameter, the CA-IDMS returns an error at runtime.

**SQL =**

Specifies the SQL syntax standard that the precompiler should apply when checking the validity of SQL statements in the program.

The precompiler issues a warning if it detects an SQL statement that does not comply with the standard specified in this parameter.

If this parameter is not specified, the default is the same as specifying SQL = NO.

**NO**

Means that compliance with a named SQL standard is not checked or enforced, and all CA-IDMS extensions are permitted.

**89**

Directs the precompiler to use ANSI X3.135-1989 (Rev), *Database Language SQL with integrity enhancement*, as the standard for compliance.

**FIPS**

Directs the precompiler to use FIPS PUB 127-1, *Database Language SQL*, as the standard for compliance.

**LIST**

Directs the precompiler to create a listing of the program with precompiler messages.

If this parameter is specified, the program listing is written to the SYSLST file.

If this parameter is not specified, the default is the same as specifying NOList.

The precompiler directive NODMLIST, included in the program source, overrides the EXEC PGM parameter LIST.

►► See Appendix C, “Precompiler Directives,” for more information on NODMLIST.

**NOList**

Directs the compiler not to create a listing of the program with precompiler messages.

The precompiler directive DMLIST, included in the program source, overrides the EXEC PGM parameter NOList.

►► See Appendix C, “Precompiler Directives,” for more information on DMLIST.

**COBOL =**

Specifies the version of COBOL with which COBOL statements generated by the precompiler must comply.

If this parameter is not specified, the default is the same as specifying COBOL = 2.

**1**

Directs the precompiler to comply with versions of COBOL that precede VS-COBOL II when generating COBOL statements.

**2**

Directs the precompiler to comply with VS-COBOL II when generating COBOL statements.

**85**

Directs the precompiler to comply with COBOL85, the version of COBOL required for the Fujitsu COBOL compilers, when generating COBOL statements.

**DATE =**

Specifies the format of the DATE data type to be used for communication between the program and the database when the access module is executed.

**TIME =**

Specifies the format of the TIME data type to be used for communication between the program and the database when the access module is executed.

**Note:** You can use the DATE and TIME parameters to override the default for the installation.

**ISO**

Specifies that the format of the DATE data type should comply with the standard of the International Standards Organization. Formats used when ISO is specified are:

Data type	Format	Example
DATE	<i>yyyy-mm-dd</i>	1990-12-15
TIME	<i>hh.mm.ss</i>	16.43.17
TIMESTAMP	<i>yyyy-mm-dd-hh.mm.ss.nnnn</i>	1990-12-15-16.43.17.123456

**USA**

Specifies that the format of the DATE data type should comply with the standard of the IBM USA standard. Formats used when USA is specified are:

Data type	Format	Example
DATE	<i>mm/dd/yyyy</i>	12/15/1990
TIME	<i>hh:mm AM</i> <i>hh:mm PM</i>	4:43 PM
TIMESTAMP	<i>yyyy-mm-dd-hh.mm.ss.nnnn</i>	1990-12-15-16.43.17.123456

**EUR**

Specifies that the format of the DATE data type should comply with the standard of the IBM European standard. Formats used when EUR is specified are:

Data type	Format	Example
DATE	<i>dd.mm.yyyy</i>	15.12.1990
TIME	<i>hh.mm.ss</i>	16.43.17
TIMESTAMP	<i>yyyy-mm-dd-hh.mm.ss.nnnn</i>	1990-12-15-16.43.17.123456

### JIS

Specifies that the format of the DATE data type should comply with the standard of the Japanese Industrial Standard Christian Era. Formats used when JIS is specified are:

Data type	Format	Example
DATE	<i>yyyy-mm-dd</i>	1990-12-15
TIME	<i>hh:mm:ss</i>	16:43:17
TIMESTAMP	<i>yyyy-mm-dd-hh.mm.ss.nnnn</i>	1990-12-15-16.43.17.123456

## 5.3 Compiling the program

**CA-IDMS precompiler:** The CA-IDMS precompiler modifies the program that you submit. CA-IDMS comments out SQL statements and substitutes calls to the DBMS. The entire source program is now in compilable form.

Here is an example of an SQL statement that has been commented out by the precompiler, and the code that the precompiler has substituted:

```
011200*      EXEC SQL
011300*          FETCH CURS1 BULK :EMPDATA
011400*          START :INDEX-CNTR ROWS :NUM-ROWS
011500*      END-EXEC.
                MOVE 4 TO SQLCLNO
                MOVE 16 TO SQLCMD
                MOVE 1 TO SQLARG
                MOVE 4 TO SQLSID
                MOVE 278 TO SQLTBL
                MOVE 6 TO SQLMRO
                MOVE INDEX-CNTR TO SQLSRO
                MOVE NUM-ROWS TO SQLNRO
                CALL 'IDMSSQL' USING
                        SQLRPB
                        SQLCA
                        SQLCA
                        SQLCIB
                        SQLPIB
                        SQLCA
                        EMPDATA
                        SQLCA
                        SQLCA
                        SQLCA
                .
```

**Language compiler:** To compile the program, you submit the source program, as successfully modified by the precompiler, to the language compiler. Output from the compiler consists of an object program and a source listing.

**Link editing:** The linkage editor edits the object program into a specified load library. Output from the linkage editor consists of a load module and a link map.

►► See Appendix A, “Sample JCL,” for JCL and further information about compiling and link editing a program.

## 5.4 Creating the access module

**About access modules:** An access module is the executable form of the SQL statements that a program issues. When you create an access module, you also invoke the optimizer. The optimizer automatically determines the most efficient access to the data requested by the SQL statements. CA-IDMS stores the access strategy in the access module.

**How you create an access module:** You create an access module with an SQL statement, CREATE ACCESS MODULE. If you accept all defaults, the access module you create:

- Is qualified with the name of the default schema for the user session
- Is stored in the DDLCATLOD area of the application dictionary to which you are connected
- Is created as version 1 if no access module of the same name and version exists in the dictionary
- Has no schema-name mapping to replace existing table or view qualifiers in SQL statements in the RCMs that the access module contains
- Is defined with AUTO RECREATE ON, which means that the DBMS will attempt to re-create the access module at runtime if a change has been made to the definition of a table accessed the module or if the RCM has been re-created since it was included in the access module
- Is defined with VALIDATE ALL, which means that the DBMS will check the definition for each table in the access module before executing the first statement in the access module
- Will execute with a default isolation of cursor stability and allow a transaction to perform updates
- Will execute with a ready mode of shared retrieval on all areas it accesses

### 5.4.1 Overriding access module defaults

**Access module name qualifier:** Qualify the access module name if you want to associate the access module with a schema that is not the default for the SQL session in which the CREATE ACCESS MODULE statement is issued.

Ownership of the schema that qualifies the access module affects authority to use the access module under CA-IDMS internal security. The owner of the schema must have authority to execute the statements in the access module, and the authorities must be grantable for another user to execute the access module.

►► For specific rules regarding schema ownership and authority to execute access modules under CA-IDMS security, refer to *CA-IDMS Security Administration*.

**Access module version number:** Specify an access module version number according to site standards.

You can use the version number of the access module to represent the version of the application that you want to execute at runtime.

►► See 5.5, “Executing the application” on page 5-16, for more information.

**Schema-name mapping for tables and views:** Supply schema-name mapping to specify a qualifier that should replace a table or view qualifier in the RCMs that the access module contains. Schema-name mapping allows you to specify the database that the access module accesses.

In this example, unqualified table and view names, and table and view names qualified with EMP\_SCH, are mapped to a schema called EMP\_TSTSCH. When the access module executes, a reference to the EMPLOYEE table or the EMP\_SCH.EMPLOYEE table will change to the EMP\_TSTSCH.EMPLOYEE table:

```
EXEC SQL
  CREATE ACCESS MODULE EMPINFO01
    FROM EMPDICT.EMPDSP01,
         EMPDICT.EMPDSP02,
         EMPDICT.EMPDSP03,
         EMPDICT.EMPADD01,
         EMPDICT.EMPUPD01,
         EMPDICT.EMPUPD02,
         EMPDICT.EMPDEL01
    MAP EMP_SCH TO EMP_TSTSCH,      ←— Schema-name mapping
    MAP NULL TO EMP_TSTSCH
END-EXEC.
```

You can subsequently change the schema-name mapping by creating a new access module or altering an existing one. This lets you change the database that the application accesses without precompiling the programs again.

►► For more information about altering an access module, see 5.4.2, “Altering an access module” on page 5-15

**Automatic access module re-creation:** At runtime, if the DBMS detects that the database definition of a table specified in the access module has changed since the access module was created, it automatically recreates the access module unless the access module was defined with AUTO RECREATE OFF.

If the AUTO RECREATE option is OFF at runtime, the DBMS returns an error with an SQLCERC value of 1014.

**Table definition timestamp validation:** The DBMS validates the definition timestamp of every table accessed by statements in the access module before executing the access module unless you specify VALIDATE BY RCM or VALIDATE BY STATEMENT. Validation failure is a condition that requires re-creation of the access module.

BY RCM causes validation only for tables accessed by statements in the RCM to be executed. BY STATEMENT causes validation only for tables accessed by the statement to be executed.

One of these specifications may be appropriate if the application contains sections of code that are infrequently executed.

**Transaction state:** The default transaction state is READ WRITE unless you specify the READ ONLY parameter. READ ONLY will cause an error to be returned at runtime attempts to perform an update. The combination of READ ONLY and a ready mode of update will cause an error when you create the access module (see **Ready mode** below).

A program can override the transaction state specified for the access module with the SET TRANSACTION statement.

SET TRANSACTION must precede most statements in the transaction. Refer to *CA-IDMS SQL Reference Guide* for detailed information.

A transaction with an isolation level of transient read is automatically a READ ONLY transaction. A specification of READ WRITE for the access module or the transaction is ignored when the isolation level of the transaction is transient read.

**Isolation level:** Specify the DEFAULT ISOLATION parameter only if cursor stability is not the appropriate isolation level for executing the application.

►► See 2.6, “Concurrency control and isolation levels” on page 2-13, for information about the effect of isolation level.

**Ready mode:** With the READY parameter, you can specify ready mode for one, some, or all areas.

Ready mode refers to the type of area lock the DBMS sets for the database transaction. The effect of the area lock differs depending on whether the execution environment is the central version or local mode. For example, for a program running under the central version, a ready mode of protected retrieval prevents concurrent transactions from updating data in the area, but for a local mode program, it does not prevent concurrent updates.

If you specify the PRECLAIM option for an area, the DBMS sets area locks on the first database access statement (to any area) in the transaction. If you do not specify PRECLAIM for an area, the default is INCREMENTAL, meaning that the area lock is set on the first access to that area.

**Default ready mode:** You should accept the default ready mode unless experience proves there is a reason to override it.

- For more information about ready mode options, refer to:
- Documentation of the CREATE ACCESS MODULE statement in the *CA-IDMS SQL Reference Guide*
  - *CA-IDMS Database Administration*

**Actual ready mode:** The actual ready mode at runtime depends on the interaction of transaction state, specified ready mode, and the status of the area (initially defined in the DMCL).

The following two tables present the actual ready mode in each possible interaction.

**READ ONLY ready modes:** This table presents the actual ready modes when the transaction state is READ ONLY:

Specified ready mode	Area status	Actual ready mode
(No specification)	Transient retrieval	Transient retrieval
	Retrieval	Shared retrieval
	Update	Shared retrieval
Any retrieval mode	Transient retrieval	Transient retrieval
	Retrieval	As specified
	Update	Shared retrieval
Any update mode	Transient retrieval	Transient retrieval
	Retrieval	Shared retrieval
	Update	Shared retrieval

**READ WRITE ready modes:** This table presents the actual ready modes when the transaction state is READ WRITE:

Specified ready mode	Area status	Actual ready mode
(No specification)	Transient retrieval	Transient retrieval
	Retrieval	Shared retrieval
	Update	Shared update
Any retrieval mode	Transient retrieval	Transient retrieval
	Retrieval	As specified
	Update	As specified
Any update mode	Transient retrieval	(Runtime error)
	Retrieval	(Runtime error)
	Update	As specified

## 5.4.2 Altering an access module

**What you can change:** With an ALTER ACCESS MODULE statement, you can change any specification that you made on the CREATE ACCESS MODULE statement. You can add, drop, or replace RCMs.

► Refer to documentation of the ALTER ACCESS MODULE statement in the *CA-IDMS SQL Reference Guide* for more information about altering an access module.

**Changing schema-name mapping:** To change the schema-name mapping for the access module, you must reprocess all RCMs by specifying the REPLACE ALL parameter, as in this example:

```
EXEC SQL
  ALTER ACCESS MODULE EMPINFO01
    REPLACE ALL
      MAP EMP SCH TO EMP PRODSCH,
      MAP NULL TO EMP_PR0DSCH
  END-EXEC.
```

## 5.5 Executing the application

**Batch jobs:** You can execute a batch job under the central version or in local mode.

JCL for executing an SQL application program in batch is presented in Appendix A, “Sample JCL.”

**SYSIDMS parameters:** In batch JCL, you can tailor certain aspects of the runtime environment by specifying SYSIDMS parameters. The following table summarizes these options:

SYSIDMS parameter	What it does
DMCL	In local mode, specifies the name of the DMCL load module to be used
DBNAME	Specifies the name of the database to be accessed at runtime if the application does not explicitly specify one
CVMACH	Specifies the name of the VM virtual machine in which the DC/UCF system is executing
SQLTRACE	Activates or deactivates the facility that traces all SQL requests made by the application

►► See the *CA-IDMS Database Administration* manual for more information about SYSIDMS parameters.

**Execution privilege:** The privileges required to access a CA-IDMS database using SQL depends on how CA-IDMS database resources are secured.

If CA-IDMS internal security is in effect, authority to access the database through the program derives from ownership of the schema that qualifies the access module name.

►► See the discussion of qualifying the access module name in 5.4.1, “Overriding access module defaults” on page 5-11, for more information.

If CA-IDMS resources are secured by an external security system, the executing user must hold appropriate privileges on all resources that the application program accesses. The schema name has no significance except as a qualifier.

►► See your security administrator for more information about privileges required to access CA-IDMS.

## 5.6 Testing the access module

**Which access module executes?**: The default access module that is executed at runtime is the access module associated with the program that issues the first SQL statement executed within the SQL session.

A program is associated with an access module when the program is precompiled.

►► See 5.2, “Precompiling the program” on page 5-4, for more information about associating a program with an access module.

There are two ways to override at runtime the access module default that is set at precompile time:

- The program issues a SET ACCESS MODULE statement before the database transaction begins
  - See 6.2.2, “Using the SET ACCESS MODULE statement” on page 6-7, for more information about using the SET ACCESS MODULE statement.
- A different version of the access module is used because a test version option has been set for the DC session in which the program is executing

**Test versions:** If there is a version of the access module that matches the test version setting, the matching version is executed. If an access module with a matching version is not found at runtime, version 1 of the access module is executed.

►► See documentation of DCUF TEST in *CA-IDMS System Tasks and Operator Commands* for more information about test versions.

## 5.7 Debugging the application

CA-IDMS provides these tools that you can use to debug the SQL portion of the application program:

- The Command Facility
- The SQL trace facility
- The EXPLAIN statement

### 5.7.1 The Command Facility

The Command Facility is a tool for a user to issue ad hoc SQL statements in an interactive online environment or in batch mode.

You can use this facility to test SQL statement syntax and to test conditions of the database both when you are designing the application and, if necessary, while debugging.

You can use CA-OLQ to access CA-IDMS with SQL. Refer to *CA-OLQ Reference Guide* for more information.

This example shows a query submitted online to the Command Facility and the result table returned. A successful SELECT statement, such as the one shown here, can be declared as a cursor with no change to the syntax.

OCF 15.0 ONLINE IDMS NO ERRORS	1/16
<pre>SELECT PROJ_ID, EST_START_DATE, PROJ_DESC FROM DEMOPROJ.PROJECT WHERE EST_START_DATE &gt; CURRENT DATE ORDER BY 2; ** ** PROJ_ID      EST_START_DATE      PROJ_DESC ** -----      -----      ----- ** C203        1998-02-01      Consumer study ** C240        1998-06-01      Service study ** C200        1999-01-15      New brand research ** D880        1999-11-01      Systems Analysis ** P634        2000-02-01      TV ads - WTVK ** P200        2000-09-01      Christmas media ** ** 6 rows processed</pre>	

► Refer to *CA-IDMS Command Facility* for more information about using the Command Facility.

## 5.7.2 SQL trace facility

You can use the SQL trace facility to trace execution of the SQL statements in a batch program.

You activate the SQL trace facility by specifying the SYSIDMS parameter SQLTRACE=ON.

In this example, the SQL trace facility reports on the SQL processing for a SELECT statement submitted through IDMSBCF, the batch Command Facility. The trace facility shows the steps in dynamically executing the SELECT, including an automatic CONNECT.

```

SELECT R.REFTABLE AS "PARENT",
       K.REFCOLUMN AS "PARENT COLUMN",
       R.NAME AS "RELATIONSHIP"
  FROM SYSTEM.CONSTRAINT R,
       SYSTEM.CONSTKEY K
 WHERE R.SCHEMA = K.SCHEMA
   AND R.NAME = K.NAME
   AND R.SCHEMA = 'REL'
   AND R.TABLE   = 'C_EMPLOYEE'
   AND R.UNIQUE >= ''
   OR R.COMPRESS <= '';
Verb=07 CONNECT TO SYSSQL
Verb=20 PREPARE--> SELECT R.REFTABLE AS "PARENT",
Verb=11 DESCRIBE
Verb=19 OPEN
Verb=16 FETCH
S Q L SQLCODE=0100  REASON CODE=0000
Verb=03 CLOSE
                                         Caller=IDMSBCF  SQLSEQ=000001 *** S Q L
                                         Caller=IDMSBCF  SQLSEQ=000008 *** S Q L
                                         Caller=IDMSBCF  SQLSEQ=000005 *** S Q L
                                         Caller=IDMSBCF  SQLSEQ=000007 *** S Q L
                                         Caller=IDMSBCF  SQLSEQ=000006 *** S Q L
                                         Caller=IDMSBCF  SQLSEQ=000002 *** S Q L

PARENT          PARENT COLUMN           RELATIONSHIP
C_DEPARTMENT    C_DEPT_ID              DEPT_EMPLOYEE
C_PROJECT       C_PROJ_ID              EMP_PROJECT

2 rows processed
Verb=05 COMMIT continue
                                         Caller=IDMSBCF  SQLSEQ=000003 *** S Q L
:
:
:
```

You can activate and deactivate the SQL trace facility within the logic of the program. You do this by issuing calls to the IDMSIN01 entry point to the IDMS module.

► See Appendix D, “Calls to IDMSIN01,” for information about the requirements for calling IDMSIN01 to activate or deactivate the SQL trace facility.

## 5.7.3 EXPLAIN statement

You can use the EXPLAIN statement to analyze the optimized access strategy for an SQL statement. An aspect of database definition or the formation of the SQL statement can result in a relatively inefficient strategy for a given SQL statement. The information produced by the EXPLAIN statement can suggest corrective measures.

► Refer to the *CA-IDMS SQL Reference Guide* for further information about the EXPLAIN statement and its use.

### 5.7.4 Online debugger

You can debug online application program execution using the CA-IDMS online debugger. The online debugger allows you to:

- Set breakpoints in the program
  - Stop execution of the program at a breakpoint
  - Examine and optionally alter conditions that exist at the breakpoint
  - Resume program execution
- Refer to *CA-IDMS Online Debugger* for more information about debugging online application programs.

# Chapter 6. SQL Programming Techniques

---

6.1 About this chapter . . . . .	6-3
6.2 Modularized programming . . . . .	6-4
6.2.1 Sharing a cursor . . . . .	6-4
6.2.2 Using the SET ACCESS MODULE statement . . . . .	6-7
6.3 Pseudoconversational programming . . . . .	6-9
6.3.1 Using SUSPEND SESSION and RESUME SESSION . . . . .	6-9
6.3.2 Scrolling through a list of rows . . . . .	6-10
6.3.3 Updating a row after a pseudoconverse . . . . .	6-11
6.4 Managing concurrent sessions . . . . .	6-15
6.4.1 Session management concepts . . . . .	6-15
6.4.2 Implementing concurrent sessions . . . . .	6-16
6.5 Creating and using a temporary table . . . . .	6-18
6.6 Bill-of-materials explosion . . . . .	6-21
6.6.1 What to do . . . . .	6-21
6.6.2 Sample program . . . . .	6-24



## 6.1 About this chapter

This chapter presents programming techniques that increase the processing capability of the program and reduce the demand for system resources. In several cases you can achieve these results because of CA-IDMS SQL extensions.

## 6.2 Modularized programming

You can design an SQL application using modularized programming techniques. CA-IDMS provides extensions to ANSI-standard SQL that allow a program to:

- Share a cursor that was opened by another program
- Specify the access module that is to be executed for the program

### 6.2.1 Sharing a cursor

**About shared cursors:** A shared cursor is declared and opened in one program and accessed in another program.

**Requirements:** These are the requirements for declaring and using a shared cursor:

- The cursor declaration in the first program must specify the GLOBAL parameter.

In this example, program EMPGET declares and opens a global cursor to select benefits information:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EMPGET.  
. . .  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
EXEC SQL  
DECLARE EMP_CRSR GLOBAL CURSOR FOR  
    SELECT EMP_ID,  
           JOB_ID,  
           SALARY_AMOUNT,  
           BONUS_PERCENT  
      FROM BENEFITS  
     WHERE EMP_ID = :EMP-ID  
END-EXEC.  
. . .  
PROCEDURE DIVISION.  
  
EXEC SQL  
    OPEN EMP_CRSR  
END-EXEC.
```

- Only the program that contains the global cursor declaration can contain the OPEN statement for the global cursor.
- A program that shares the cursor must make an external cursor declaration.

In this example, program EMPUPD declares an external cursor to share the global cursor declared in EMPGET:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EMPUPD.  
. .  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
EXEC SQL  
    DECLARE EMP_CRSR EXTERNAL CURSOR  
END-EXEC.
```

- Any number of programs that execute within the same database transaction can share a global cursor.
- All programs that share a cursor must be part of the same access module.

The GLOBAL parameter is not valid for cursors associated with dynamically-compiled SELECT statements.

**Verifying external cursors:** The precompiler does not verify the validity of a DECLARE EXTERNAL CURSOR statement. The programmer has responsibility verifying that programs meet the requirements for declaring and accessing a global cursor.

**Shared cursor example:** In this example, EMPGET declares EMP\_CRSR as an updateable global cursor, opens the cursor, and fetches the row. After checking the results of the fetch, EMPGET passes control to EMPUPD. EMPUPD declares EMP\_CRSR as an external cursor and performs a positioned update using input values for the updateable columns.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EMPGET.  
. . .  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
EXEC SQL  
    DECLARE EMP_CRSR GLOBAL CURSOR FOR  
        SELECT EMP_ID,  
               JOB_ID,  
               SALARY_AMOUNT,  
               BONUS_PERCENT  
          FROM BENEFITS  
         WHERE EMP_ID = :EMP-ID  
           FOR UPDATE OF SALARY_AMOUNT,  
                     BONUS_PERCENT  
END-EXEC.  
. . .  
  
PROCEDURE DIVISION.  
  
EXEC SQL  
    OPEN EMP_CRSR  
END-EXEC.  
  
PERFORM FETCH-ROUTINE UNTIL END-FETCH='Y'  
  
FETCH-ROUTINE.  
  
EXEC SQL  
    FETCH EMP_CRSR  
      INTO :EMP-ID,  
            :JOB-ID,  
            :SALARY-AMOUNT INDICATOR SALARY-AMOUNT-I,  
            :BONUS-PERCENT INDICATOR BONUS-PERCENT-I  
END-EXEC.  
  
IF SQLCODE = 100 MOVE 'Y' TO END-FETCH.  
IF SALARY-AMOUNT-I = -1 OR BONUS-PERCENT-I = -1  
    PERFORM INITIALIZE-NULLED-VARIABLES.  
  
CALL EMPUPD.
```

---

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EMPUPD.
.
.
.
DATA DIVISION.
WORKING-STORAGE SECTION.

EXEC SQL
  DECLARE EMP_CRSR EXTERNAL CURSOR
END-EXEC.

.
.
.

PROCEDURE DIVISION.

.
.
.

MOVE INPUT-SALARY-AMOUNT TO SALARY-AMOUNT.
MOVE INPUT-BONUS-PERCENT TO BONUS-PERCENT.

EXEC SQL
  UPDATE BENEFITS
    SET SALARY_AMOUNT = :SALARY-AMOUNT,
        BONUS_PERCENT = :BONUS-PERCENT
      WHERE CURRENT OF EMP_CRSR
END-EXEC.

```

## 6.2.2 Using the SET ACCESS MODULE statement

**Why you use it:** You use a SET ACCESS MODULE statement to specify in the program what access module should be executed for a database transaction. SET ACCESS MODULE overrides the default access module specification for the duration of the transaction.

**Default access module specification:** The default access module specification is the one associated with the program that initiates the SQL session — that is, the first program to issue an SQL statement.

► See 5.4, “Creating the access module” on page 5-11, for information about how an access module is associated with a program.

The default access module is the access module that is executed unless the program issues a SET ACCESS MODULE statement. The SET ACCESS MODULE specification remains in effect until the database transaction ends. After the database transaction ends, the default access module is re-established.

**When to issue SET ACCESS MODULE:** The SET ACCESS MODULE statement is valid only if the program issues it in the transaction before it issues an SQL statement requesting dictionary or database access.

► Refer to the *CA-IDMS SQL Reference Guide* for the list of statements that can precede SET ACCESS MODULE in a database transaction.

**Using a host variable:** You can specify the access module name in a host variable on the SET ACCESS MODULE. This allows the specification of an access module to be decided by conditions not known until runtime.

**Tip:** When you define a host variable for the access module name, an eight-byte character field suffices because an access module name is limited to eight characters.

**SET ACCESS MODULE example:** In this example, program EMPACT declares a global cursor and issues a SET ACCESS MODULE statement before starting a transaction with an OPEN statement:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.  EMPACT.  
. . .  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
EXEC SQL  
  DECLARE EMP_CRSR GLOBAL CURSOR FOR  
    SELECT EMP_ID  
      FROM EMPLOYEE  
      WHERE STATUS = 'A'  
END-EXEC.  
. . .  
  
PROCEDURE DIVISION.  
  
MOVE 'EMPAPPL3' TO AM-NAME.  
  
EXEC SQL  
  SET ACCESS MODULE :AM-NAME  
END-EXEC.  
  
EXEC SQL  
  OPEN EMP_CRSR  
END-EXEC.  
. . .
```

## 6.3 Pseudoconversational programming

**About pseudoconversational programming:** Pseudoconversational programming is an online programming technique that frees certain resources while the system waits for a response from the online user. This permits an online environment to support more concurrent processing by conserving limited resources such as storage pool and program pool space.

To facilitate pseudoconversational programming in an SQL application, CA-IDMS supports the SUSPEND SESSION and RESUME SESSION statements.

**Updating after a pseudoconverse:** The online user's response may call for modification of data that was retrieved by the program. This section discusses techniques for updating after a pseudoconverse, including consideration of whether the program needs to verify that the data has not changed since it was retrieved.

### 6.3.1 Using SUSPEND SESSION and RESUME SESSION

**What SUSPEND SESSION does:** When the program issues a SUSPEND SESSION statement, the DBMS releases all resources associated with the SQL session *except* those needed to resume the current session and transaction:

- The database connection
- Cursor currencies
- Locks held by any currently active transaction
- Temporary tables
- Dynamically prepared SQL statements

SUSPEND SESSION does not cause a commit or rollback of work.

**What RESUME SESSION does:** RESUME SESSION reestablishes the active SQL session and database transaction. All characteristics and cursor positions of the session and transaction are restored to what they were when the program issued the SUSPEND SESSION statement.

In a pseudoconversational program, RESUME SESSION must be the first SQL statement the application issues after a SUSPEND SESSION statement.

**Advantages of suspending and resuming:** Since a suspended session preserves database transaction and SQL session characteristics, you can use SUSPEND SESSION and RESUME SESSION in these types of applications:

- Scrolling through a list of result rows
- Updating a row with user input

The following sections discusses how to use SUSPEND SESSION and RESUME SESSION in these types of processing.

### 6.3.2 Scrolling through a list of rows

**Retrieval list using bulk fetch:** You can use a bulk fetch and a suspended session to develop an online application for scrolling through a list of rows. Each fetch statement retrieves a screen display of rows. The session is suspended before the pseudoconverse and resumed when the user requests the next set of rows to display. Since the DBMS has maintained cursor position during the suspended session, the next execution of the fetch statement automatically retrieves the next set of rows in the cursor result table.

**Retrieval list example:** In this example, having already declared a host variable array with as many occurrences as there are rows in a screen display, the program declares and opens the POSITION\_CRSR cursor to retrieve data about employees by department:

```
EXEC SQL
  DECLARE POSITION_CRSR CURSOR FOR
    SELECT P.EMP_ID,
           E.DEPT_ID,
           P.JOB_ID,
           P.SALARY_AMOUNT,
      FROM POSITION_P, EMPLOYEE E
     WHERE P.EMP_ID = E.EMP_ID
       AND E.DEPT_ID = :DEPT-ID
END-EXEC.

EXEC SQL
  OPEN POSITION_CRSR
END-EXEC.
```

The program then iterates the following logic until the online user exits this thread of the application. The first fetch uses the value of INPUT-DEPT-ID. The second fetch retrieves the next set of employees for the department because the DBMS has maintained the cursor position during the suspended session:

```

EXEC SQL
  FETCH POSITION_CRSR
    BULK :BULK-POSITION
END-EXEC.

IF SQLCODE = 100 MOVE 'Y' TO END-FETCH.

EXEC SQL
  SUSPEND SESSION
END-EXEC.

(Move retrieved values to display fields)

MAP OUT ...

(Pseudoconverse)

MAP IN ...

EXEC SQL
  RESUME SESSION
END-EXEC.

IF END-FETCH = 'Y' ... ← Close cursor and either
select a new department or exit

```

**Scrolling backwards:** Scrolling backwards through an online retrieval list requires pageable map processing. If necessary, you can manage pageable map processing by using:

- The CA-IDMS scratch area and scratch management statements to temporarily store and re-access retrieved data
  - CA-ADS pageable mapping in a CA-ADS application
- See the applicable CA-IDMS program language reference manual for more information about scratch area management.

### 6.3.3 Updating a row after a pseudoconverse

**Using an updateable cursor:** During a suspended session, the DBMS maintains the cursor position of an open cursor and also the lock on the current cursor row. This means that a program running under the cursor stability isolation level can resume the suspended session and perform a positioned update without checking whether the row has been updated by a concurrent database transaction.

**Updateable cursor example:** In this example, the program fetches a row from the BENEFITS\_CRSR cursor, suspends the session, and displays the row to the online user. Following user input, the program resumes the session and performs a positioned update with user input:

```

EXEC SQL
DECLARE BENEFITS_CRSR FOR
  SELECT JOB_ID,
         SALARY_AMOUNT,
         BONUS_PERCENT
    FROM BENEFITS
   WHERE EMP_ID = :EMP-ID
END-EXEC.

EXEC SQL
  OPEN BENEFITS_CRSR
END-EXEC.

EXEC SQL
  FETCH BENEFITS_CRSR
    INTO :JOB_ID,
          :SALARY_AMOUNT,
          :BONUS_PERCENT
END-EXEC.

EXEC SQL
  SUSPEND SESSION
END-EXEC.

(Move retrieved values to display fields)

MAP OUT ...

(Pseudoconverse)

MAP IN...

(Program moves input data to host variables)

EXEC SQL
  RESUME SESSION
END-EXEC.

EXEC SQL
  UPDATE BENEFITS
    SET SALARY_AMOUNT = :SALARY-AMOUNT,
        BONUS_PERCENT = :BONUS-PERCENT
   WHERE CURRENT OF BENEFITS_CRSR
END-EXEC.

EXEC SQL
  COMMIT
END-EXEC.

```

**Searched update after a pseudoconverse:** When a database transaction running under the default isolation mode of cursor stability suspends the session, the DBMS releases any lock it set on the base row(s) of a single-row SELECT result. No locks are maintained on rows resulting from bulk selects in this situation, and only the lock on the last row fetched in a bulk fetch is maintained under cursor stability during a suspended session.

This means that a concurrent database transaction can update the data retrieved by a single-row SELECT statement or FETCH BULK statement while the session of the original transaction is suspended. In these situations, the program should check

whether the data has been modified since it was retrieved before applying an update after the pseudoconverse.

**Checking whether the row was modified:** To be able to check whether a row has been modified, your processing environment can create and maintain a column for a last-update timestamp value. An alternative is to compare the values of all fields to be updated with the values that were retrieved.

**Maintaining a last-update timestamp:** To maintain a last-update timestamp for a table row, use these procedures:

1. Define a last-update column for each table with data type TIMESTAMP and NOT NULL WITH DEFAULT
2. In the program, define the host variable for the last-update timestamp column as a character field with length 26
3. Set the last-update timestamp column to the value of the special register CURRENT TIMESTAMP when modifying the row

You can add a last-update column to an existing table using the ALTER TABLE statement. Refer to the *CA-IDMS SQL Reference Guide* for information about ALTER TABLE.

**How you check the row before updating:** To determine whether a row has been modified since the program retrieved it, you attempt a searched update with a search condition that includes a comparison to verify that the last-update timestamp value has not changed.

**Searched update example:** In this example, the program issues a single-row SELECT statement from the POSITION table using the primary key of the table. The program suspends the SQL session and displays the retrieved row to the online user:

```
MOVE MAP-EMP-ID TO EMP-ID.  
MOVE MAP-JOB-ID TO JOB-ID.  
  
EXEC SQL  
  SELECT EMP_ID,  
        JOB_ID,  
        SALARY_AMOUNT,  
        LAST_UPDATED  
    INTO :EMP-ID,  
         :JOB-ID,  
         :SALARY-AMOUNT,  
         :LAST-UPDATED  
   FROM POSITION  
 WHERE EMP_ID = :EMP-ID  
END-EXEC.  
  
EXEC SQL  
  SUSPEND SESSION  
END-EXEC.  
.  
.  
.  
MAP OUT ...
```

**(Pseudoconverse)**

Following the pseudoconverse, the program issues an update to the single row using input from the online user. The update executes only if the row has not been modified since it was retrieved:

```
MAP IN ...  
  
MOVE MAP-SALARY-AMOUNT TO SALARY-AMOUNT.  
  
EXEC SQL  
  RESUME SESSION  
END-EXEC.  
  
EXEC SQL  
  UPDATE POSITION  
    SET SALARY_AMOUNT = :SALARY-AMOUNT,  
        LAST_UPDATED = CURRENT TIMESTAMP  
  WHERE EMP_ID = :EMP-ID  
    AND JOB_ID = :JOB-ID  
    AND LAST_UPDATED = :LAST-UPDATED  
END-EXEC.  
  
IF SQLCODE = 100 PERFORM ROW-CHANGED.
```

## 6.4 Managing concurrent sessions

The ability to maintain concurrent active sessions allows the program to access multiple databases with parallel database transactions. For example, one session can retrieve data from one database and, using that data, perform an update operation on another database.

**CAUTION:**

If an application attempts to access the *same* database in concurrent sessions, there is an inherent risk of deadlock.

### 6.4.1 Session management concepts

**Concurrent session identifier:** When a session begins, CA-IDMS assigns an identifier to the session and maintains the session identifier internally. All SQL statements implicitly reference the session identifier during execution.

If there are multiple concurrent sessions, each session has its own session ID. To manage multiple sessions, an application must manipulate the session identifier directly.

**Data declaration requirements:** To manipulate the session identifier, the program must first:

- Declare one host variable of usage SQLSESS
- Define a variable in working storage for each of the multiple sessions that the program will maintain

When the program begins an SQL session, CA-IDMS returns the session identifier to the SQLSESS host variable that the program has defined. The program must save the SQLSESS value of each concurrent session.

**How CA-IDMS uses the SQLSESS variable:** If the program declares an SQLSESS host variable, all calls to CA-IDMS pass the SQLSESS host variable as a parameter to indicate the session to which the SQL statements should be directed.

CA-IDMS does not alter the session ID value in this parameter unless the statement being executed terminates the session (that is, on a COMMIT, RELEASE, or ROLLBACK RELEASE). If the session is terminated, CA-IDMS initializes the SQLSESS host variable.

**What the program must do:** Before executing an SQL statement, the application must ensure that the correct session ID value has been moved to the SQLSESS host variable.

## 6.4.2 Implementing concurrent sessions

**Declaring the SQLSESS host variable:** To implement concurrent sessions, the program must declare a host variable to which CA-IDMS assigns the session identifier of the active SQL session:

```
EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC.

01  IDMS-SESS-ID    USAGE SQLSESS.

EXEC SQL
  END DECLARE SECTION
END-EXEC.
```

**Saving the session ID value:** The precompiler expands the SQLSESS host variable to an 8-byte character field. Therefore, to save session ID values, the application program must define work fields that also are 8-byte character fields:

```
01  WS-SESSION-IDS.
  05 SESS1-ID          PIC X(8).
  05 SESS2-ID          PIC X(8).
```

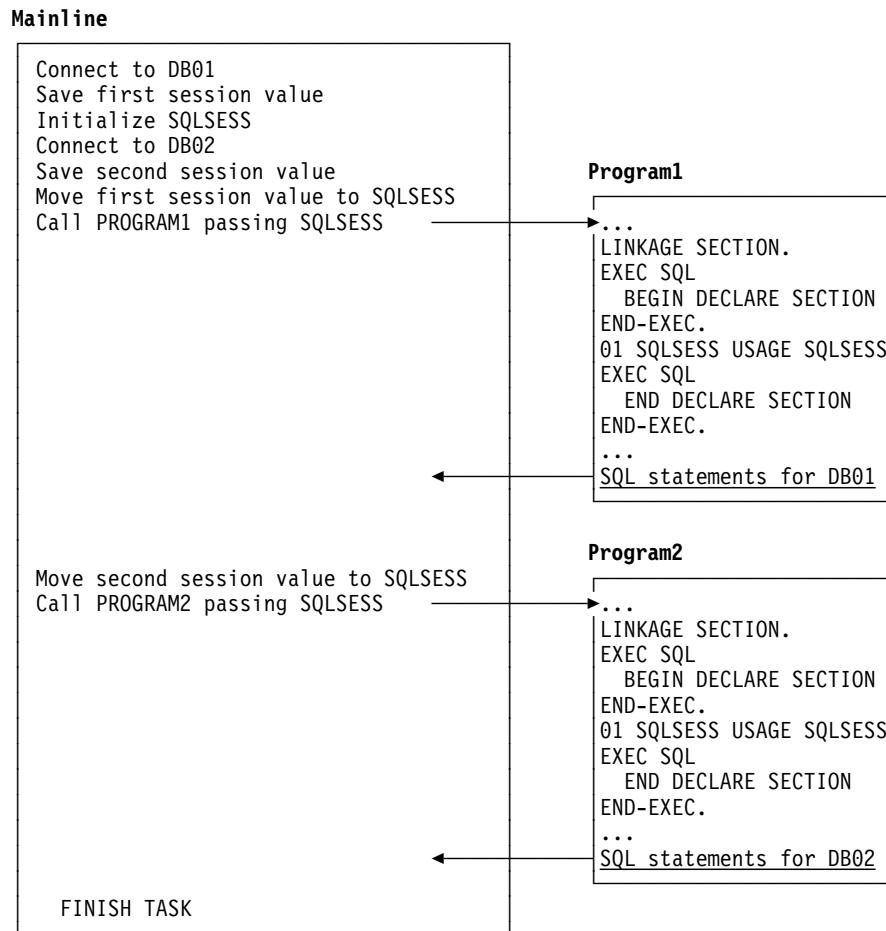
**Multiple session steps:** These are the steps in a typical scenario for managing multiple sessions:

1. Begin a session accessing Database 1
2. Move IDMS-SESS-ID to SESS1-ID
3. Initialize IDMS-SESS-ID by moving spaces to it
4. CONNECT TO Database 2
5. Move IDMS-SESS-ID to SESS2-ID

At this point, the current session ID value is the one representing the second session. To make the first session the current session, the application program would move the value in SESS1-ID to IDMS-SESS-ID.

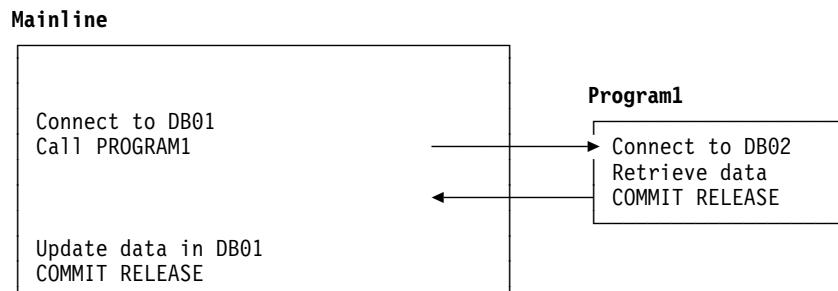
**Multiple sessions started by one program:** The following diagram illustrates a scenario in which a program manages session IDs to maintain multiple concurrent sessions.

In this case, the mainline program initiates both sessions and passes the appropriate session ID to each subordinate program to indicate which session the subprogram should process. Each subprogram must also declare a session identifier to hold the value passed from the mainline program.



**Multiple sessions started by different programs:** The following diagram illustrates a scenario in which multiple sessions are begun by multiple programs.

In this case, Program 1 must declare a session ID to indicate that a separate session is desired; otherwise, the CONNECT statement will return an error. However, no manipulation of the session ID is required.



## 6.5 Creating and using a temporary table

**About temporary tables:** A temporary table differs from a database table in these ways:

- A temporary table exists only as long as the database transaction in which it is created
- You cannot create an index on a temporary table
- A temporary table cannot be referenced in a view or a referential constraint
- A temporary table cannot be accessed by another database transaction

With the above exceptions, a program can access a temporary table and manipulate temporary table data as it does with a database table.

**Why use a temporary table?:** A temporary table can be useful for certain processing requirements, such as to:

- Take a snapshot of information in the database
- Avoid re-accessing base tables multiple times to retrieve the same information, to process efficiently and assure that the information does not change
- Perform certain operations that cannot be done with a single SQL statement, such as inserting rows into a table using data retrieved from the same table

**Caution using a temporary table:** Since you cannot create an index on a temporary table, access to a temporary table is always serial. Accessing data in a temporary table with many rows may degrade the performance of the program.

**How you create a temporary table:** You create a temporary table in the procedural section of the program by issuing a CREATE TEMPORARY TABLE statement. This statement requires:

- A temporary table name
- Column names
- Column definitions

CA-IDMS maintains temporary tables in the scratch area. The program does not supply information about the physical characteristics of a temporary table.

► Refer to the *CA-IDMS SQL Reference Guide* for further information about creating temporary tables in particular. Refer to the *CA-IDMS Database Administration* for further information about creating tables in general.

**Naming a temporary table:** When you create a temporary table, you should name it in a way that cannot match the name of any table or view that may be created. If a temporary table name matches the name of a base table or view, the optimizer will

assume the name refers to the base table or view, and the temporary table will not be accessed.

**Cursor for a temporary table:** The program can declare a cursor for a temporary table. However, when you create the access module for the program, the optimizer issues a warning in response to any reference to the temporary table other than in the CREATE TEMPORARY TABLE statement.

Thus, the programmer has the responsibility of verifying that the cursor declaration and the CREATE TEMPORARY TABLE statement are compatible.

**Temporary table example:** In this example, the program creates a temporary table of manager names and ids using information in the EMPLOYEE table. (The EMPLOYEE table itself associates the id of a manager with the name of the subordinate employee, not the name of the manager.) Using a cursor, the program accesses a row of the temporary table and selects employees from the EMPLOYEE table who report to the manager identified in the temporary table row.

This is the cursor declaration and the statement to create the temporary table:

WORKING STORAGE SECTION.

```
EXEC SQL
DECLARE TEMP_CRSR CURSOR FOR
    SELECT *
        FROM TEMP_MGR
        ORDER BY 3
END EXEC
.
.
.
PROCEDURE DIVISION.

EXEC SQL
CREATE TEMPORARY TABLE TEMP_MGR
    (TEMP_MGR_ID           INTEGER,
     TEMP_FNAME            CHAR(20),
     TEMP_LNAME            CHAR(20))
END-EXEC.
```

This statement adds manager information to the temporary table:

```
EXEC SQL
    INSERT INTO TEMP_MGR
    SELECT DISTINCT E.MANAGER_ID,
               M.EMP_FNAME,
               M.EMP_LNAME
        FROM EMPLOYEE E, EMPLOYEE M
       WHERE E.MANAGER_ID = M.EMP_ID
END-EXEC.
```

This statement establishes a current cursor row for the temporary table:

```
EXEC SQL
  FETCH TEMP_CRSR
  INTO :MGR-ID,
       :MGR-FNAME,
       :MGR-LNAME
END-EXEC.
```

This statement performs a bulk select of employees who report to the manager in the current cursor row. Depending on processing requirements, this statement could be a bulk fetch:

```
EXEC SQL
  SELECT EMP-FNAME,
         EMP-LNAME,
         DEPT-ID
  BULK :BULK-EMPLOYEE
  FROM EMPLOYEE
  WHERE MANAGER-ID = :MGR-ID
    AND TERMINATION-DATE IS NULL
END-EXEC.
```

## 6.6 Bill-of-materials explosion

This section presents a sample program that performs a bill-of-materials explosion. A discussion of the concepts involved precedes the sample program.

### 6.6.1 What to do

**Maximum level:** The sample program establishes a value of 100 as the limit of levels for the explosion in its use of the MAX-LEVELS variable. A limit of 100 is for illustration only; a program can set a higher or lower limit.

```
01 LIMITS-AND-CONSTANTS.
  02 NUMBER-OF-CURSORS  PIC S9      COMP VALUE 3.
  02 MAX-LEVELS        PIC S9(4)   COMP VALUE 100.
  02 NULL-KEY-VALUE    PIC 9(7)    VALUE 0.
```

**Cursor declarations:** The program declares three different cursors with identical definitions. The cursor issues a join of the PART and COMPONENT tables that produces a result table of component parts for each part.

```
EXEC SQL DECLARE CURSOR1 CURSOR FOR
  SELECT COMPONENT_PART,
         QUANTITY,
         PART_NAME
    FROM COMPONENT C,
         PART P
   WHERE C.PART = :CURRENT-KEY
     AND C.COMPONENT_PART > :PREVIOUS-COMPONENT
     AND P.NUMBER = C.PART
  ORDER BY COMPONENT_PART
END-EXEC

EXEC SQL DECLARE CURSOR2 CURSOR FOR
  SELECT COMPONENT_PART,
         QUANTITY,
         PART_NAME
    FROM COMPONENT C,
         PART P
   WHERE C.PART = :CURRENT-KEY
.
.
.
```

The minimum number of cursors needed is two. Theoretically, the program could declare more cursors with identical definitions, up to a number of cursors equal to the maximum level for the explosion. However, for most bill-of-material explosions, it is more practical and efficient to add program logic that allows the three cursors to be reused as illustrated in the sample program later in this section.

**Getting the first row:** The GET-FIRST-ROW section of the program issues a single-row select from the PART table. The search condition equates an input part number (TOP-KEY), the part to be exploded, with PART\_NUMBER, the unique key of the PART table.

This select verifies the existence of the part and also retrieves its name.

```
EXEC SQL
  SELECT PART_NUMBER, PART_NAME
  INTO :CURRENT-KEY, :COMPONENT-NAME
  FROM PART
  WHERE PART_NUMBER = :TOP-KEY
END-EXEC.
```

**Going to the first level:** In the FETCH-NEXT-ROW section, the program opens a cursor to retrieve the component parts that make up the current part, whose number it has assigned to CURRENT-KEY. The program fetches the first row of the cursor result table.

```
FETCH-NEXT-ROW SECTION.
  PERFORM OPEN-CURRENT-CURSOR.

  IF CURRENT-CURSOR = 1
    EXEC SQL
      FETCH CURSOR1 INTO
        :COMPONENT-KEY, :QTY, :COMPONENT-NAME
    END-EXEC
  ELSE IF CURRENT-CURSOR = 2
  .
  .
  .
```

**Going down more levels:** If the first fetch succeeds, the program executes the DOWN-ONE-LEVEL section. In this section, the program:

- Assigns the part number in the first row fetched to CURRENT-KEY
- Increments the current level by 1
- Increments the current cursor by 1 if the current cursor is less than 3

Because the program reuses the three cursors, it attempts to close a cursor in the CLOSE-CURRENT-CURSOR section before it opens the cursor in the OPEN-CURRENT-CURSOR section. For the first three levels of the explosion, the DBMS will ignore the CLOSE statement because the specified cursor has not yet been opened.

Using the part number retrieved in the fetch by the previous cursor, the program now fetches the first component part of the next level down by opening the current cursor and fetching from it. This logic is repeated until a fetch returns an SQLCODE of 100 (in effect, no more levels) or the defined maximum level is reached.

**Saved keys:** Each time it goes down a level, the program saves the part number used in the fetch:

```
DOWN-ONE-LEVEL SECTION.
  IF CURRENT-LEVEL > MAX-LEVELS
    NEXT SENTENCE
  ELSE
    MOVE COMPONENT-KEY TO CURRENT-KEY
    MOVE COMPONENT-KEY TO SAVE-KEY (CURRENT-LEVEL)
  .
  .
```

By saving the key, the program can later retrieve the part number for a level and execute the backup logic described below.

**When there are no more levels:** When there are no more levels, the program executes the BACKUP-ONE-LEVEL section. It subtracts 1 from the level number and retrieves the saved keys for the current and previous levels.

```
BACKUP-ONE-LEVEL SECTION.  
    SUBTRACT 1 FROM CURRENT-LEVEL.  
    IF CURRENT-LEVEL > 0  
        MOVE SAVE-KEY (CURRENT-LEVEL) TO PREVIOUS-COMPONENT.  
    IF CURRENT-LEVEL > 1  
        MOVE SAVE-KEY (CURRENT-LEVEL - 1) TO CURRENT-KEY  
    .  
    .  
    .
```

Since the cursor result tables are ordered by component part number and one of the conditions of each is C.COMPONENT\_PART > :PREVIOUS-COMPONENT, the program re-establishes cursor position in the list of components by limiting the rows selected to those not yet processed. Each time a cursor is re-opened, the first row of the result table is the next component to be processed,

This allows the program both to reuse a cursor and to fetch the next row for the previous level.

**Completing the explosion:** The process of going down a level until there are no more levels, going back one level, and attempting to go down again is repeated until backing up reaches the top level. This means the bill-of-materials explosion is complete.

## 6.6.2 Sample program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    EXPLODE.  
  
ENVIRONMENT DIVISION.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 SQLMSGs.  
  02 SQLMMAX          PIC S9(8) COMP VALUE +6.  
  02 SQLMSIZE          PIC S9(8) COMP VALUE +80.  
  02 SQLMCNT           PIC S9(8) COMP.  
  02 SQLMLINE          OCCURS 6 TIMES PIC X(80).  
  
01 REQ-WK.  
  02 REQUEST-CODE      PIC S9(8) COMP.  
  02 REQUEST-RETURN     PIC S9(8) COMP.  
  
01 LIMITS-AND-CONSTANTS.  
  02 NUMBER-OF-CURSORS  PIC S9      COMP VALUE 3.  
  02 MAX-LEVELS         PIC S9(4)  COMP VALUE 100.  
  02 NULL-KEY-VALUE     PIC 9(7)   VALUE 0.  
  
01 CURSOR-FLAGS.  
  02 CURSOR-FLAG        OCCURS 3 TIMES PIC X.  
  
01 KEY-TABLE.  
  02 SAVE-KEY          OCCURS 100 TIMES PIC 9(7).  
  
01 WORK-FIELDS.  
  02 CURRENT-LEVEL      PIC S9(4) COMP.  
  02 CURRENT-CURSOR     PIC S9(4) COMP.  
  02 DISPLAY-LEVEL       PIC ZZ9.  
  02 WARNING-MSG         PIC X(40).  
  02 SQLVALUE            PIC ----9.  
  
EXEC SQL    BEGIN DECLARE SECTION      END-EXEC  
01 DBNAME                PIC X(8).  
01 PREVIOUS-COMPONENT    PIC S9(7)  COMP-3.  
01 TOP-KEY                PIC S9(7)  COMP-3.  
01 CURRENT-ROW.  
  02 CURRENT-KEY          PIC S9(7)  COMP-3.  
  02 COMPONENT-KEY        PIC S9(7)  COMP-3.  
  02 QTY                   PIC S9(5)V99 COMP-3.  
  02 COMPONENT-NAME       PIC X(30).  
EXEC SQL    END    DECLARE SECTION      END-EXEC.
```

```
*****
      DECLARE CURSORS
*****

EXEC SQL  DECLARE CURSOR1 CURSOR FOR
    SELECT COMPONENT_PART,
           QUANTITY,
           PART_NAME
      FROM COMPONENT C,
           PART P
     WHERE C.PART = :CURRENT-KEY
       AND C.COMPONENT_PART > :PREVIOUS-COMPONENT
       AND P.NUMBER = C.PART
    ORDER BY COMPONENT_PART
END-EXEC

EXEC SQL  DECLARE CURSOR2 CURSOR FOR
    SELECT COMPONENT_PART,
           QUANTITY,
           PART_NAME
      FROM COMPONENT C,
           PART P
     WHERE C.PART = :CURRENT-KEY
       AND C.COMPONENT_PART > :PREVIOUS-COMPONENT
       AND P.NUMBER = C.PART
    ORDER BY COMPONENT_PART
END-EXEC

EXEC SQL  DECLARE CURSOR3 CURSOR FOR
    SELECT COMPONENT_PART,
           QUANTITY,
           PART_NAME
      FROM COMPONENT C,
           PART P
     WHERE C.PART = :CURRENT-KEY
       AND C.COMPONENT_PART > :PREVIOUS-COMPONENT
       AND P.NUMBER = C.PART
    ORDER BY COMPONENT_PART
END-EXEC
*****


PROCEDURE DIVISION.

EXEC SQL
  WHENEVER SQLERROR GO TO SQL-ERROR
END-EXEC.
```

```
MAINLINE SECTION.  
    ACCEPT DBNAME.  
    ACCEPT TOP-KEY.  
*  INITIALIZE VARIABLES TO GET US STARTED  
    MOVE 1 TO CURRENT-LEVEL.  
    MOVE 1 TO CURRENT-CURSOR.  
    MOVE SPACES TO CURSOR-FLAGS.  
    MOVE NULL-KEY-VALUE TO PREVIOUS-COMPONENT.  
*  
    PERFORM GET-FIRST-ROW.  
    PERFORM FETCH-NEXT-ROW  
        UNTIL CURRENT-LEVEL = 0.  
    EXEC SQL COMMIT RELEASE END-EXEC.  
    GOBACK.  
  
GET-FIRST-ROW SECTION.  
    EXEC SQL CONNECT TO :DBNAME           END-EXEC.  
    EXEC SQL  
        SELECT PART_NUMBER, PART_NAME  
        INTO :CURRENT-KEY, :COMPONENT-NAME  
        FROM PART  
        WHERE PART_NUMBER = :TOP-KEY  
    END-EXEC.  
  
    IF SQLCODE = 100  
        MOVE 0 TO CURRENT-LEVEL  
        DISPLAY '***** INVALID PART NUMBER: '  
                TOP-KEY  
    ELSE  
        DISPLAY '***** BILL OF MATERIALS FOR '  
                'PART: ' CURRENT-KEY ' '  
                COMPONENT-NAME ' *****'  
        DISPLAY '*****'  
                '*****'  
                '*****'.  
*****
```

```
FETCH-NEXT-ROW SECTION.  
  PERFORM OPEN-CURRENT-CURSOR.  
  
  IF      CURRENT-CURSOR = 1  
    EXEC SQL  
      FETCH CURSOR1 INTO  
        :COMPONENT-KEY, :QTY, :COMPONENT-NAME  
    END-EXEC  
  ELSE IF CURRENT-CURSOR = 2  
    EXEC SQL  
      FETCH CURSOR2 INTO  
        :COMPONENT-KEY, :QTY, :COMPONENT-NAME  
    END-EXEC  
  ELSE IF CURRENT-CURSOR = 3  
    EXEC SQL  
      FETCH CURSOR3 INTO  
        :COMPONENT-KEY, :QTY, :COMPONENT-NAME  
    END-EXEC.  
  
  IF SQLCODE = 100  
    PERFORM BACKUP-ONE-LEVEL  
  ELSE  
    PERFORM PRINT-CURRENT-ROW  
    PERFORM DOWN-ONE-LEVEL.  
  
OPEN-CURRENT-CURSOR SECTION.  
  IF CURSOR-FLAG (CURRENT-CURSOR) NOT = '0'  
    MOVE '0' TO CURSOR-FLAG (CURRENT-CURSOR)  
    IF      CURRENT-CURSOR = 1  
      EXEC SQL  
        OPEN CURSOR1  
    END-EXEC  
    ELSE IF CURRENT-CURSOR = 2  
      EXEC SQL  
        OPEN CURSOR2  
    END-EXEC  
    ELSE IF CURRENT-CURSOR = 3  
      EXEC SQL  
        OPEN CURSOR3  
    END-EXEC.
```

```
CLOSE-CURRENT-CURSOR SECTION.  
  IF CURRENT-CURSOR-FLAG (CURRENT-CURSOR) = '0'  
    MOVE ' ' TO CURRENT-CURSOR-FLAG (CURRENT-CURSOR)  
    IF      CURRENT-CURSOR = 1  
      EXEC SQL  
        CLOSE CURSOR1  
      END-EXEC  
    ELSE IF CURRENT-CURSOR = 2  
      EXEC SQL  
        CLOSE CURSOR2  
      END-EXEC  
    ELSE IF CURRENT-CURSOR = 3  
      EXEC SQL  
        CLOSE CURSOR3  
      END-EXEC.  
  
DOWN-ONE-LEVEL SECTION.  
  IF CURRENT-LEVEL > MAX-LEVELS  
    NEXT SENTENCE  
  ELSE  
    MOVE COMPONENT-KEY TO CURRENT-KEY  
    MOVE COMPONENT-KEY TO SAVE-KEY (CURRENT-LEVEL)  
    MOVE NULL-KEY-VALUE TO PREVIOUS-COMPONENT  
    ADD 1 TO CURRENT-LEVEL  
    IF CURRENT-CURSOR = MAX-CURSORS  
      MOVE 1 TO CURRENT-CURSOR  
      PERFORM CLOSE-CURRENT-CURSOR  
    ELSE  
      ADD 1 TO CURRENT-CURSOR  
      PERFORM CLOSE-CURRENT-CURSOR.  
  
BACKUP-ONE-LEVEL SECTION.  
  SUBTRACT 1 FROM CURRENT-LEVEL.  
  IF CURRENT-LEVEL > 0  
    MOVE SAVE-KEY (CURRENT-LEVEL) TO PREVIOUS-COMPONENT.  
  IF CURRENT-LEVEL > 1  
    MOVE SAVE-KEY (CURRENT-LEVEL - 1) TO CURRENT-KEY  
  ELSE  
    MOVE TOP-KEY TO CURRENT-KEY.  
  PERFORM CLOSE-CURRENT-CURSOR.  
  IF CURRENT-CURSOR = 1  
    MOVE MAX-CURSORS TO CURRENT-CURSOR  
  ELSE  
    SUBTRACT 1 FROM CURRENT-CURSOR.
```

```
PRINT-CURRENT-ROW SECTION.  
  MOVE CURRENT-LEVEL TO DISPLAY-LEVEL.  
  IF CURRENT-LEVEL > MAX-LEVELS  
    MOVE 'MAXIMUM LEVEL, COMPONENTS NOT LISTED'  
      TO WARNING-MSG  
  ELSE  
    MOVE SPACES TO WARNING-MSG.  
  DISPLAY '  DISPLAY-LEVEL  
  |  PART:  COMPONENT-KEY  
  |  |  COMPONENT-NAME  
  |  QTY:  QTY  
  |  |  WARNING-MSG.  
SQL-ERROR SECTION.  
  DISPLAY '***** ERROR IN SQL STATEMENT'  
  |  *****.  
  DISPLAY 'PROGRAM          SQLPGM  
DISPLAY 'COMPILED        SQLDATE  
MOVE SQLCLNO TO SQLVALUE.  
DISPLAY 'SQL LINE NUMBER  SQLVALUE  
MOVE SQLCODE TO SQLVALUE.  
DISPLAY 'SQLCODE          SQLVALUE  
MOVE SQLCERC TO SQLVALUE.  
DISPLAY 'REASON CODE     SQLVALUE  
MOVE SQLCERC TO SQLVALUE.  
DISPLAY 'ERROR CODE       SQLVALUE  
MOVE SQLCNRP TO SQLVALUE.  
DISPLAY 'ROWS PROCESSED   SQLVALUE  
  
MOVE 4 TO REQUEST-CODE.  
CALL 'IDMSIN01' USING SQLRPB, REQ=WK,  
      SQLCA, SQLMSGS.  
IF REQUEST-RETURN NOT = 4  
  MOVE 1 TO LINE-CNT  
  PERFORM DISP=MSG UNTIL LINE-CNT > SQLMCNT.  
  
DISP-MSG SECTION.  
  DISPLAY SQLMLINE (LINE-CNT).  
  ADD 1 TO LINE-CNT.
```



# Chapter 7. Using Dynamic SQL

---

7.1 About this chapter . . . . .	7-3
7.2 About dynamic SQL . . . . .	7-4
7.3 Dynamic insert, update, and delete operations . . . . .	7-6
7.3.1 Using EXECUTE IMMEDIATE . . . . .	7-6
7.3.2 Using PREPARE . . . . .	7-7
7.3.3 Using EXECUTE . . . . .	7-9
7.4 Executing prepared SELECT statements . . . . .	7-10
7.4.1 What to do . . . . .	7-10
7.4.2 Sample program . . . . .	7-11
7.5 Executing prepared CALL statements . . . . .	7-16
7.5.1 What to do . . . . .	7-16
7.5.2 Sample program . . . . .	7-18



## 7.1 About this chapter

Dynamic SQL refers to an SQL statement that is not known to the program at precompile time and therefore is compiled dynamically when the program executes. CA-IDMS provides dynamic SQL to allow the program to formulate, compile, and execute a DML statement at runtime.

This chapter explains how to implement dynamic SQL, depending on the processing requirement of the program and the capabilities of the programming language.

## 7.2 About dynamic SQL

**To insert, update, or delete:** You implement dynamic SQL with a small set of SQL statements. For SQL DML other than SELECT or CALL, these statements are:

- EXECUTE IMMEDIATE — Dynamically compiles and executes the statement
- PREPARE — Dynamically compiles the statement
- EXECUTE — Executes a prepared statement

If the statement to be dynamically compiled could be issued more than once in the program, you should use the combination of PREPARE and EXECUTE statements.

**To select:** To dynamically compile and execute a SELECT statement, you take these steps:

1. Formulate the statement
2. Prepare the statement and optionally describe the result table to CA-IDMS
3. Declare or allocate a cursor using the dynamically compiled SELECT statement

**To CALL an external routine:** To dynamically compile and execute a CALL statement, you take these steps:

1. Formulate the statement
2. Prepare the statement and optionally describe the result table to CA-IDMS
3. Declare or allocate a cursor using the dynamically compiled CALL statement

**Host language dependency:** If the number and type of columns in a dynamic SELECT or CALL are not known at compile time, the host language must provide explicit support for dynamic storage allocation because the variable storage requirements for the data to be retrieved can be derived only from information returned to the SQLDA when the SELECT statement is prepared.

**No host variables:** A dynamic SQL statement that is prepared or executed using an EXECUTE IMMEDIATE statement cannot reference host variables within the text of the statement. If you want to repeatedly execute a statement, such as an UPDATE, using different update values each time, you must use dynamic parameters in place of host variables.

►► Refer to the *CA-IDMS SQL Reference Guide* for more information on dynamic parameters.

**Precompiling with NOINSTALL:** A program that consists entirely of dynamic SQL statements, session and transaction management statements, requires no RCM. Therefore, you may precompile such a program with the NOINSTALL option. This directs the precompiler to check syntax and not to store an RCM, thus eliminating the need for updating the dictionary. If, however, SQL requests will be issued from more than one program within a single transaction, each such program must have its RCM

included in the access module being used. This requirement holds, regardless of whether all of the statements within a program are dynamic or not. Therefore, as general practice, you should avoid specifying the NO INSTALL option.

## 7.3 Dynamic insert, update, and delete operations

You can perform a dynamic insert, update, or delete using EXECUTE or EXECUTE IMMEDIATE. EXECUTE is valid only when the statement has been dynamically compiled with a PREPARE statement.

### 7.3.1 Using EXECUTE IMMEDIATE

**When to use it:** Use EXECUTE IMMEDIATE to dynamically compile and execute a statement that will be issued only once in the transaction.

If a program consists mainly of dynamic SQL statements, consider using EXECUTE IMMEDIATE for the few remaining SQL statements. You can precompile the program with the NOINSTALL option, eliminating an RCM and an access module to execute the program. This may be more efficient in your processing environment.

**EXECUTE IMMEDIATE example:** In this example, the program builds an INSERT statement in working storage and moves the complete statement to a host variable, STATEMENT-TEXT. The program issues an EXECUTE IMMEDIATE statement on the text contained in the host variable:

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01  INSERT-STATEMENT-TEXT.  
02 FILLER          PIC X(21)  VALUE  
    "INSERT INTO C_DIVISION VALUES ('".  
02 DIV-CODE-TEXT   PIC X(3).  
02 FILLER          PIC X(3)  VALUE  
    "','".  
02 DIV-NAME-TEXT   PIC X(40).  
02 FILLER          PIC X(2)  VALUE  
    "','".  
02 DIV-HEAD-ID-TEXT PIC X(4).  
02 FILLER          PIC X(3)  VALUE  
    ")".  
  
EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
01 STATEMENT-TEXT  PIC X(76).  
EXEC SQL END DECLARE SECTION  END-EXEC.  
  
PROCEDURE DIVISION.  
  
MOVE INPUT-DIV-CODE TO DIV-CODE-TEXT.  
MOVE INPUT-DIV-NAME TO DIV-NAME-TEXT.  
MOVE INPUT-DIV-HEAD-ID TO DIV-HEAD-ID-TEXT.  
MOVE INSERT-STATEMENT-TEXT TO STATEMENT-TEXT.  
  
EXEC SQL  
EXECUTE IMMEDIATE :STATEMENT-TEXT  
END-EXEC.
```

**Error-checking:** There is no error-checking technique that is specific to EXECUTE IMMEDIATE. Check for SQLCODE < 0, or check for a specific SQLSTATE value if appropriate.

### 7.3.2 Using PREPARE

**Why you use PREPARE:** You use the PREPARE statement to dynamically compile an SQL statement that is formulated at runtime. You should prepare the statement if:

- The statement may be issued more than once during a transaction
- The statement may be a SELECT

**Determining information about the prepared statement:** You can use either the DESCRIBE option of the PREPARE statement or a separate DESCRIBE statement to determine the following information:

- Whether or not the prepared statement is a SELECT
- If the prepared statement is a SELECT, the number of result columns to be returned and the name and format of each of the result columns
- The format of any dynamic parameters that must be supplied as input values when the statement is executed or an associated cursor is opened

In order to retrieve this information, you must allocate at least one SQL descriptor area. You need to allocate two descriptor areas if you want to retrieve information about both result columns and dynamic parameters.

**Note:** Descriptor areas must be defined using the SQLDA structure.

**Declaring SQLDA:** The program can declare the default descriptor area SQLDA with an INCLUDE statement:

```
EXEC SQL
  INCLUDE SQLDA
    NUMBER OF COLUMNS 20
END-EXEC.
```

**Declaring SQLDA in CA-ADS:** If you are using descriptor areas in CA-ADS, you can create a work record layout through IDD as described in the *CA-ADS User Guide*. This work record must match the SQLDA layout and the initial values should conform to the data types.

The following displays the CA-ADS format of the SQLDA:

```
01  SQLDA.  
05  SQLDAID      PIC X(8).  
05  SQLN         PIC S9(9) COMP  
                  VALUE +n .  
05  SQLD         PIC S9(9) COMP.  
05  SQLVAR        OCCURS n.  
    10  SQLLEN      PIC S9(9) COMP.  
    10  SQLTYPE     PIC S9(4) COMP.  
    10  SQLSCALE    PIC S9(4) COMP.  
    10  SQLPRECISION PIC S9(4) COMP.  
    10  SQLALN      PIC S9(4) COMP.  
    10  SQLNALN    PIC S9(4) COMP.  
    10  SQLNULL     PIC S9(4) COMP.  
    10  SQLNAME     PIC X(32).
```

where n is the maximum number of occurrences of SQLVAR

**SQLDA values:** An SQL descriptor area used to retrieve information about the output of the prepared statement contains the following values:

The value in SQLD indicates whether the statement is:

- A SELECT statement if the value is greater than 0
- Not a SELECT statement if the value is equal to 0

If greater than 0, SQLD is the number of columns in the result table of the SELECT statement.

The value in SQLN indicates the maximum number of columns the descriptor area can describe:

- The number specified in the NUMBER OF COLUMNS parameter of the INCLUDE statement
- If SQLD is greater than SQLN, the descriptor area is too small to describe the result table.

SQLVAR is a structure that occurs SQLN times. Each occurrence contains information about a result column.

►► Refer to the *CA-IDMS SQL Reference Guide* for more information.

**PREPARE example:** In this example, the program has formulated an SQL statement and has moved the character string into the host variable STATEMENT-STRING:

```
EXEC SQL  
  PREPARE DYNAMIC_STATEMENT  
    FROM :STATEMENT-STRING  
    DESCRIBE INTO SQLDA  
END-EXEC.
```

**Error-checking:** If a PREPARE statement fails to execute at runtime, CA-IDMS returns a negative value to SQLCODE.

If the SQLCODE value is -4, there may be a syntax error in the statement. If there is, the offset within the statement at which the syntax error occurred is returned to the SQLCSER field of the SQLCA.

### 7.3.3 Using EXECUTE

**Why you use EXECUTE:** You use EXECUTE to execute a dynamically compiled (prepared) statement other than SELECT. This is the format of the EXECUTE statement:

```
EXEC SQL
    EXECUTE statement-name
END-EXEC.
```

The parameter *statement-name* must correspond to the value in the same parameter of a PREPARE statement that has already been issued in the same transaction.

**EXECUTE example:** In this example, the statement prepared in an earlier example is executed:

```
EXEC SQL
    EXECUTE DYNAMIC_STATEMENT
END-EXEC.
```

**Error-checking:** There is no error-checking technique that is specific to EXECUTE. Check for SQLCODE < 0, or check for a specific SQLSTATE value if appropriate.

**Repeating EXECUTE:** You can repeat an EXECUTE statement in the same transaction because CA-IDMS retains all dynamically compiled statements for the duration of the transaction.

If the program prepares more than one statement in a database transaction using the same statement name, an EXECUTE issued for the statement name will execute the most recently prepared statement.

## 7.4 Executing prepared SELECT statements

This section presents a sample program that prepares a SELECT statement and executes it dynamically. A discussion of the concepts involved precedes the sample program.

### 7.4.1 What to do

**Declaring a cursor:** To execute a prepared SELECT statement, the program must first declare a cursor for the prepared statement.

The sample program declares this cursor:

```
EXEC SQL
    DECLARE CURSOR1 CURSOR FOR SELECT_STATEMENT
END-EXEC.
```

**Preparing the statement:** Before opening a cursor defined with a dynamic SQL statement, the program must prepare the statement.

The sample program issues this PREPARE statement:

```
EXEC SQL
    PREPARE SELECT_STATEMENT FROM :STATEMENT-TEXT
END-EXEC.
```

**Building the statement text:** In the sample program, the host variable STATEMENT-TEXT contains a character string consisting of a fixed portion of the statement to which input text is added when the program executes.

The fixed portion of the statement specifies table and columns from which data is selected. This part of the statement is initialized in working storage:

```
01 FIRST-PART-OF-STATEMENT.
02 FILLER          PIC X(32)  VALUE
  'SELECT EMP_ID, EMP_FNAME,'.
02 FILLER          PIC X(32)  VALUE
  ' EMP_LNAME, DEPT_ID,'.
02 FILLER          PIC X(32)  VALUE
  ' MANAGER_ID, START_DATE '.
02 FILLER          PIC X(32)  VALUE
  ' FROM DEMO.EMPL_VIEW_1 '.
```

The variable portion of the statement, which can specify additional selection criteria such as an ORDER BY or a WHERE clause, is completed when BUILD-SQL-STATEMENT section of the program executes.

**Declaring a host variable array:** The sample program performs a bulk fetch after it opens the cursor. The bulk fetch requires a host variable array to receive the data.

The sample program declares the host variable array within an SQL declaration section using this INCLUDE statement:

```
01  FETCH-BUFFER.  
EXEC SQL  
    INCLUDE TABLE DEMO.EMPL_VIEW_1  
        (EMP_ID, EMP_FNAME, EMP_LNAME,  
         DEPT_ID, MANAGER_ID, START_DATE)  
    NUMBER OF ROWS 50  
    LEVEL 02  
END-EXEC.
```

**Executing the fetch:** After the program builds the statement text, prepares the statement, and opens the cursor, it issues the bulk fetch:

```
FETCH-ROWS SECTION.  
EXEC SQL  
    FETCH CURSOR1  
    BULK :FETCH-BUFFER  
END-EXEC.  
MOVE 1 TO ROW-CTR.  
PERFORM DISPLAY-ROW  
    UNTIL ROW-CTR > SQLCNRP.
```

#### 7.4.2 Sample program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      EMPVIEW1.  
  
ENVIRONMENT DIVISION.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 SQLMSG$.  
  02 SQLMMAX          PIC S9(8) COMP VALUE +6.  
  02 SQLMSIZE         PIC S9(8) COMP VALUE +80.  
  02 SQLMCNT          PIC S9(8) COMP.  
  02 SQLMLINE         OCCURS 6 TIMES PIC X(80).  
  
01 REQ-WK.  
  02 REQUEST-CODE    PIC S9(8) COMP.  
  02 REQUEST-RETURN   PIC S9(8) COMP.  
01 LIMITS-AND-CONSTANTS.  
  02 MAX-TEXT-LINES   PIC S9  COMP VALUE 5.  
  
01 FIRST-PART-OF-STATEMENT.  
  02 FILLER           PIC X(32)  VALUE  
    'SELECT EMP_ID, EMP_FNAME,'.  
  02 FILLER           PIC X(32)  VALUE  
    'EMP_LNAME, DEPT_ID,'.  
  02 FILLER           PIC X(32)  VALUE  
    'MANAGER_ID, START_DATE'.  
  02 FILLER           PIC X(32)  VALUE  
    'FROM DEMO.EMPL_VIEW_1'.  
  
01 HEADING-LINE.  
  02 FILLER           PIC X(31)  VALUE  
    'ID #  FIRST NAME '.  
  02 FILLER           PIC X(23)  VALUE  
    'LAST NAME '.  
  02 FILLER           PIC X(31)  VALUE  
    'DEPT  MGR        START DATE'.  
01 DETAIL-LINE.  
  02 EMP-ID            PIC 9(5).  
  02 FILLER           PIC X(3)   VALUE SPACES.  
  02 EMP-FNAME         PIC X(20).  
  02 FILLER           PIC X(3)   VALUE SPACES.  
  02 EMP-LNAME         PIC X(20).  
  02 FILLER           PIC X(3)   VALUE SPACES.  
  02 DEPT-ID           PIC 9(5).  
  02 FILLER           PIC X(3)   VALUE SPACES.  
  02 MANAGER-ID        PIC 9(5).  
  02 FILLER           PIC X(3)   VALUE SPACES.  
  02 START-DATE        PIC X(10).
```

```

01 WORK-FIELDS.
  02 ROW-CTR          PIC S99 COMP.
  02 TEXT-CTR          PIC S99 COMP.
  02 INPUT-LINE.
    03 END-CHAR        PIC X.
      88 END-STATEMENT  VALUE ';' .
    03 FILLER           PIC X(79).
  02 SQLVALUE          PIC ____9.

01 STATEMENT-TXT2.
  02 FIXED-PART        PIC X(128).
  02 VARIABLE-PART.
    03 TEXT-LINES OCCURS 5 TIMES PIC X(80).
  77 DBNAME             PIC X(8).
01 STATEMENT-TEXT       PIC X(641).
01 FETCH-BUFFER.

EXEC SQL
  INCLUDE TABLE DEMO.EMPL_VIEW_1
    (EMP_ID, EMP_FNAME, EMP_LNAME,
     DEPT_ID, MANAGER_ID, START_DATE)
  NUMBER OF ROWS 50
  LEVEL 02
END-EXEC.

EXEC SQL     END     DECLARE SECTION          END-EXEC

*****          DECLARE CURSORS          *****
EXEC SQL
  DECLARE CURSOR1 CURSOR FOR SELECT_STATEMENT
END-EXEC
*****


PROCEDURE DIVISION.

EXEC SQL
  WHENEVER SQLERROR GO TO SQL-ERROR
END-EXEC.

MAINLINE SECTION.
  ACCEPT DBNAME.
  MOVE FIRST-PART-OF-STATEMENT TO FIXED-PART.
  MOVE 1 TO TEXT-CTR.
  PERFORM BUILD-SQL-STATEMENT
    UNTIL TEXT-CTR > MAX-TEXT-LINES.
  IF END-STATEMENT
    PERFORM PREPARE-AND-OPEN-CURSOR
    PERFORM FETCH-ROWS
      UNTIL SQLCODE = 100
    EXEC SQL COMMIT RELEASE   END-EXEC.
  GOBACK.

```

```
BUILD-SQL-STATEMENT SECTION.  
  IF NOT END-STATEMENT  
    ACCEPT INPUT-LINE  
    DISPLAY INPUT-LINE.  
  IF NOT END-STATEMENT  
    MOVE INPUT-LINE TO TEXT-LINE (TEXT-CTR)  
  ELSE  
    MOVE SPACES      TO TEXT-LINE (TEXT-CTR).  
    ADD 1 TO TEXT-CTR.  
  
PREPARE-AND-OPEN-CURSOR SECTION.  
  EXEC SQL           — CONNECT TO DATABASE  
    CONNECT TO :DBNAME  
  END-EXEC.  
  
  EXEC SQL           — SET ISOLATION MODE  
    SET TRANSACTION TRANSIENT READ  
  END-EXEC.  
  
  MOVE STATEMENT-TXT2 TO STATEMENT-TEXT.  
  EXEC SQL           — PREPARE THE SELECT  
    PREPARE SELECT_STATEMENT FROM :STATEMENT-TEXT  
  END-EXEC.  
  
  EXEC SQL           — OPEN THE CURSOR  
    OPEN CURSOR1  
  END-EXEC.  
  
  DISPLAY ' '.  
  DISPLAY ' '.  
  DISPLAY HEADING-LINE.  
  DISPLAY ' '.  
  
FETCH-ROWS SECTION.  
  EXEC SQL  
    FETCH CURSOR1  
    BULK :FETCH-BUFFER  
  END-EXEC.  
  MOVE 1 TO ROW-CTR.  
  PERFORM DISPLAY-ROW  
    UNTIL ROW-CTR > SQLCNRP.  
  
DISPLAY-ROW SECTION.  
  MOVE CORRESPONDING EMPL-VIEW-1 (ROW-CTR) TO DETAIL-LINE.  
  DISPLAY DETAIL-LINE.  
  ADD 1 TO ROW-CTR.
```

```
SQL-ERROR      SECTION.  
    DISPLAY '***** ERROR IN SQL STATEMENT'  
          ' *****'.  
    DISPLAY 'PROGRAM           ' SQLPGM  
    DISPLAY 'COMPILED          ' SQLDTS  
    MOVE SQLCLNO TO SQLVALUE.  
    DISPLAY 'SQL LINE NUMBER   ' SQLVALUE  
    MOVE SQLCODE TO SQLVALUE.  
    DISPLAY 'SQLCODE          ' SQLVALUE  
    MOVE SQLCERC  TO SQLVALUE.  
    DISPLAY 'REASON CODE       ' SQLVALUE  
    MOVE SQLCERC  TO SQLVALUE.  
    DISPLAY 'ERROR CODE        ' SQLVALUE  
    MOVE SQLCNRP  TO SQLVALUE.  
    DISPLAY 'ROWS PROCESSED    ' SQLVALUE  
  
    MOVE 4 TO REQUEST-CODE.  
    CALL 'IDMSIN01' USING SQLRPB, REQ-WK,  
          SQLCA, SQLMSG.  
    IF REQUEST-RETURN NOT = 4  
        MOVE 1 TO LINE-CNT  
        PERFORM DISP-MSG UNTIL LINE-CNT > SQLMCNT  
  
DISP-MSG SECTION.  
    DISPLAY SQLMLINE (LINE-CNT).  
    ADD 1 TO LINE-CNT.
```

## 7.5 Executing prepared CALL statements

This section presents a sample program that prepares a CALL statement and executes it dynamically. A discussion of the concepts involved precedes the sample program.

### 7.5.1 What to do

**Declaring a cursor:** To execute a prepared CALL statement, the program must first declare a cursor for the prepared statement. The sample program declares this cursor:

```
EXEC SQL
    DECLARE CURSOR1 CURSOR FOR CALL_STATEMENT
END-EXEC.
```

**Preparing the statement:** Before opening a cursor defined with a dynamic SQL statement, the program must prepare the statement. The sample program issues this PREPARE statement:

```
EXEC SQL
    PREPARE CALL_STATEMENT FROM :STATEMENT-TEXT
END-EXEC.
```

**Building the statement text::** In the sample program, the host variable STATEMENT-TEXT contains a character string consisting of a fixed portion of the statement to which input text is added when the program executes.

The fixed portion of the statement specifies the CALL statement. This part of the statement is initialized in working storage:

```
01 FIRST-PART-OF-STATEMENT.
    02 FILLER PIC X(8) VALUE 'CALL '.
```

The variable portion of the statement, which specifies the procedure-reference in the form of [schema].procedure [parameters], is completed when BUILD-SQL-STATEMENT section of the program executes.

**Declaring host variables for 3 parameters:** The sample program performs a fetch into 3 host variables after it opens the cursor.

The sample program declares following host variables within an SQL declaration :

```
01 DETAIL-LINE.
    02 P1          PIC 9(10).
    02 FILLER      PIC X(3)  VALUE SPACES.
    02 P2          PIC 9(10).
    02 FILLER      PIC X(3)  VALUE SPACES.
    02 P3          PIC X(32) VALUE SPACES.
    02 FILLER      PIC X(3)  VALUE SPACES.
01 DBNAME        PIC X(8).

01 STATEMENT-TEXT PIC X(641).
```

**Executing the fetch:** After the program builds the statement text, prepares the statement, and opens the cursor, it issues the fetch:

FETCH-ROWS SECTION.

```
EXEC SQL
    FETCH CURSOR1 INTO :P1,
                      :P2,
                      :P3
END-EXEC.
```

```
MOVE 1 TO ROW-CTR.
PERFORM DISPLAY-ROW UNTIL
    ROW-CTR > SQLCNRP.
```

## 7.5.2 Sample program

```

EXEC SQL BEGIN DECLARE SECTION          END-EXEC
01 DETAIL-LINE.
  02 P1          PIC 9(10).
  02 FILLER      PIC X(3)  VALUE SPACES.
  02 P2          PIC 9(10).
  02 FILLER      PIC X(3)  VALUE SPACES.
  02 P3          PIC X(32) VALUE SPACES.
  02 FILLER      PIC X(3)  VALUE SPACES.

01 DBNAME          PIC X(8).
01 STATEMENT-TEXT  PIC X(641).

EXEC SQL END   DECLARE SECTION          END-EXEC

01 WORK-FIELDS.
  02 ROW-CTR        PIC S99 COMP.
  02 TEXT-CTR        PIC S99 COMP.
  02 INPUT-LINE.
    03 END-CHAR      PIC X.
    88 END-STATEMENT  VALUE ';' .
    03 FILLER        PIC X(79).
  02 SQLVALUE        PIC ----9.

01 STATEMENT-TXT2.
  02 FIXED-PART     PIC X(8).
  02 VARIABLE-PART.
    03 TEXT-LINES OCCURS 5 TIMES PIC X(80).

*****
*****      DECLARE CURSORS           *****
****

EXEC SQL
  DECLARE CURSOR1 CURSOR FOR CALL_STATEMENT
END-EXEC
*****


PROCEDURE DIVISION.

EXEC SQL
  WHENEVER SQLERROR GO TO SQL-ERROR
END-EXEC.

MAINLINE SECTION.
  ACCEPT DBNAME.
  MOVE FIRST-PART-OF-STATEMENT TO FIXED-PART.
  MOVE 1 TO TEXT-CTR.
  PERFORM BUILD-SQL-STATEMENT
    UNTIL TEXT-CTR > MAX-TEXT-LINES.
  IF END-STATEMENT
    PERFORM PREPARE-AND-OPEN-CURSOR
    PERFORM FETCH-ROWS
      UNTIL SQLCODE = 100
    EXEC SQL COMMIT RELEASE   END-EXEC.
  GOBACK.

```

```
BUILD-SQL-STATEMENT SECTION.  
  IF NOT END-STATEMENT  
    ACCEPT INPUT-LINE  
    DISPLAY INPUT-LINE.  
  IF NOT END-STATEMENT  
    MOVE INPUT-LINE TO TEXT-LINES(TEXT-CTR)  
  ELSE  
    MOVE SPACES      TO TEXT-LINES(TEXT-CTR).  
  ADD 1 TO TEXT-CTR.  
  
PREPARE-AND-OPEN-CURSOR SECTION.  
  EXEC SQL          -- CONNECT TO DATABASE  
    CONNECT TO :DBNAME  
  END-EXEC.  
  
  EXEC SQL          -- SET ISOLATION MODE  
    SET TRANSACTION TRANSIENT READ  
  END-EXEC.  
  
  MOVE STATEMENT-TXT2 TO STATEMENT-TEXT.  
  EXEC SQL          -- PREPARE THE CALL  
    PREPARE CALL_STATEMENT FROM :STATEMENT-TEXT  
  END-EXEC.  
  
  EXEC SQL          -- OPEN THE CURSOR  
    OPEN CURSOR1  
  END-EXEC.  
  
  DISPLAY ' '.  
  DISPLAY ' '.  
  DISPLAY HEADING-LINE.  
  DISPLAY ' '.  
  
FETCH-ROWS SECTION.  
  EXEC SQL  
    FETCH CURSOR1  
      INTO :P1, :P2, :P3  
  END-EXEC.  
  MOVE 1 TO ROW-CTR.  
  PERFORM DISPLAY-ROW  
    UNTIL ROW-CTR > SQLCNRP.  
  
DISPLAY-ROW SECTION.  
  DISPLAY DETAIL-LINE.  
  ADD 1 TO ROW-CTR.
```

```
SQL-ERROR      SECTION.  
DISPLAY '***** ERROR IN SQL STATEMENT'  
'*****'.  
DISPLAY 'PROGRAM          ' SQLPGM  
DISPLAY 'COMPILED         ' SQLDTS  
MOVE SQLCLNO TO SQLVALUE.  
DISPLAY 'SQL LINE NUMBER  ' SQLVALUE  
MOVE SQLCODE TO SQLVALUE.  
DISPLAY 'SQLCODE          ' SQLVALUE  
MOVE SQLCERC TO SQLVALUE.  
DISPLAY 'REASON CODE     ' SQLVALUE  
MOVE SQLCERC TO SQLVALUE.  
DISPLAY 'ERROR CODE       ' SQLVALUE  
MOVE SQLCNRP TO SQLVALUE.  
DISPLAY 'ROWS PROCESSED   ' SQLVALUE  
  
MOVE 4 TO REQUEST-CODE.  
CALL 'IDMSIN01' USING SQLRPB, REQ-WK,  
      SQLCA, SQLMSG.  
IF REQUEST-RETURN NOT = 4  
  MOVE 1 TO LINE-CNT  
  PERFORM DISP-MSG UNTIL LINE-CNT > SQLMCNT.  
  
DISP-MSG SECTION.  
DISPLAY SQLMLINE (LINE-CNT).  
ADD 1 TO LINE-CNT.
```



## **Appendix A. Sample JCL**

---

A.1	About this appendix . . . . .	A-3
A.2	OS/390 . . . . .	A-4
A.3	VSE/ESA . . . . .	A-10
A.3.1	Usage . . . . .	A-12
A.4	VM/ESA . . . . .	A-14
A.4.1	Usage . . . . .	A-16
A.5	BS2000/OSD . . . . .	A-19



## A.1 About this appendix

This appendix provides sample JCL or commands for executing the precompile, access module creation, compile, and link edit steps on four operating systems:

- OS/390
- VSE/ESA
- VM/ESA
- BS2000/OSD

## A.2 OS/390

**About the examples:** The two sample JCL streams that follow contains the steps required to make a host language source program with embedded SQL into the form of executable modules. The first example is for execution under the central version, and the second example is for execution in local mode.

The host language for the examples is COBOL 1. Change the specification of precompiler name, precompiler options, and compiler name according to the host language and version of your program.

Following the second example is a table that gives the meaning of variables used in the examples.

### Central version JCL:

```
//*****  
//**          PRECOMPILE COBOL PROGRAM           **  
//*****  
//precomp EXEC PGM=IDMSDMLC,REGION=1024K,  
//          PARM='optional precompiler parameters'  
//STEPLIB  DD  DSN=idms.dba.loadlib,DISP=SHR  
//          DD  DSN=idms.loadlib,DISP=SHR  
//sysctl  DD  DSN=idms.sysctl,DISP=SHR  
//dcmsg    DD  DSN=idms.sysmsg.dd1dcmsg,DISP=SHR  
//SYS001   DD  UNIT=disk,SPACE=(TRK,(10,10))  
//SYS002   DD  UNIT=disk,SPACE=(TRK,(10,10))  
//SYS003   DD  UNIT=disk,SPACE=(TRK,(10,10))  
//SYSPCH   DD  DSN=&&source,DISP=(NEW,PASS),  
//          UNIT=disk,SPACE=(TRK,(10,5),RLSE),  
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)  
//SYSLST   DD  SYSOUT=A  
//SYSIDMS  DD  *  
DMCL=dmc1-name  
DICTNAME=dictionary-name  
Additional SYSIDMS parameters, as appropriate
```

```

/*
//SYSIPT DD *
Host language source statements with embedded SQL
/*
//***** CREATE ACCESS MODULE *****
//*****
//acmod EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
//SYSIPT DD *
CREATE ACCESS MODULE statement ;
COMMIT WORK RELEASE ;
/*
//***** COMPILE COBOL PROGRAM *****
//*****
//compile EXEC PGM=IKFCBL00,REGION=240K,
// PARM='DECK,NOLOAD,NOLIB,BUF=500000,SIZE=150K'
//STEPLIB DD DSN=sys1.cobol.linklib,DISP=SHR
//SYSUT1 DD UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT2 DD UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT3 DD UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT4 DD UNIT=disk,SPACE=(TRK,(10,5))
//SPUNCH DD DSN=&&object,DISP=(NEW,PASS),
// UNIT=disk,SPACE=(TRK,(10,5),RLSE),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSN=&&source,DISP=(OLD,DELETE)

```

```

//***** LINK PROGRAM MODULE *****
//*****
//link EXEC PGM=IEWL,REGION=300K,PARM='LET,LIST,XREF'
//SYSUT1 DD UNIT=disk,SPACE=(TRK,(20,5))
//SYSLIB DD DSN=sys1.coblib,DISP=SHR
//loadlib DD DSN=idms.loadlib,DISP=SHR
//SYSLMOD DD DSN=user.loadlib,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&&object,DISP=(OLD,DELETE)
// DD *
INCLUDE loadlib(IDMS)           ← Non-CICS only
INCLUDE loadlib(IDMSCINT)        ← CICS only
ENTRY userentry
NAME userprog(R)
/*
//*

```

**Local mode JCL:**

```

//***** PRECOMPILE COBOL PROGRAM ****
//**                                         PRECOMPILE COBOL PROGRAM      **
//*****                                         *****                         ****
//precomp EXEC PGM=IDMSDMLC,REGION=1024K,
//          PARM='precompiler parameters'
//STEPLIB  DD  DSN=idms.dba.loadlib,DISP=SHR
//          DD  DSN=idms.loadlib,DISP=SHR
//dictb   DD  DSN=idms.appldict.ddldml,DISP=SHR
//dloddb   DD  DSN=idms.appldict.ddldcld,DISP=SHR
//sqldd   DD  DSN=idms.syssql.ddlcat,DISP=SHR
//sqlxdd  DD  DSN=idms.syssql.ddlcatx,DISP=SHR
//sqllod  DD  DSN=idms.syssql.ddlcatl,DISP=SHR
//dcmsg   DD  DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//sysjrn1 DD  DSN=idms.tapejrn1,DISP=(NEW,CATLG,UNIT=tape
//SYS001  DD  UNIT=disk,SPACE=(TRK,(10,10))
//SYS002  DD  UNIT=disk,SPACE=(TRK,(10,10))
//SYS003  DD  UNIT=disk,SPACE=(TRK,(10,10))
//SYSPCH  DD  DSN=&&source,DISP=(NEW,PASS,DELETE),
//          UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSLST  DD  SYSOUT=A
//SYSIDMS DD  *
DMCL=dmc1-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
//SYSIPT  DD  *
Host language source statements with embedded SQL
/*
//***** CREATE ACCESS MODULE ****
//*****                                         *****                         ****
//**                                         CREATE ACCESS MODULE      **
//*****                                         *****                         ****
//accmod EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB  DD  DSN=idms.dba.loadlib,DISP=SHR
//          DD  DSN=idms.loadlib,DISP=SHR
//dictb   DD  DSN=idms.appldict.ddldml,DISP=SHR
//dloddb   DD  DSN=idms.appldict.ddldcld,DISP=SHR
//sqldd   DD  DSN=idms.syssql.ddlcat,DISP=SHR
//sqlxdd  DD  DSN=idms.syssql.ddlcatx,DISP=SHR
//sqllod  DD  DSN=idms.syssql.ddlcatl,DISP=SHR
//dcmsg   DD  DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//sysjrn1 DD  DSN=idms.tapejrn1,DISP=(NEW,CATLG,UNIT=tape
//SYSLST  DD  SYSOUT=A
//SYSIDMS DD  *
DMCL=dmc1-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
//SYSIPT  DD  *
CREATE ACCESS MODULE statement ;
COMMIT WORK RELEASE ;
/*

```

```

//*****COMPILER*****                                         ****
//**          COMPILE COBOL PROGRAM                         **
//*****LINKER*****                                         ****
//compile EXEC PGM=IKFCBL00,REGION=240K,
//           PARM='DECK,NOLOAD,NOLIB,BUF=500000,SIZE=150K'
//STEPLIB  DD  DSN=sys1.cobol.linklib,DISP=SHR
//SYSUT1  DD  UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT2  DD  UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT3  DD  UNIT=disk,SPACE=(TRK,(10,5))
//SYSUT4  DD  UNIT=disk,SPACE=(TRK,(10,5))
//SYSPUNCH DD  DSN=&&object,DISP=(NEW,PASS,DELETE),
//           UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD  SYSOUT=A
//SYSIN   DD  DSN=&&source,DISP=(OLD,DELETE)
//*****LINKER*****                                         ****
//**          LINK PROGRAM MODULE                         **
//*****LOADLIB*****                                         ****
//link    EXEC PGM=IEWL,REGION=300K,PARM='LET,LIST,XREF'
//SYSUT1  DD  UNIT=disk,SPACE=(TRK,(20,5))
//SYSLIB  DD  DSN=sys1.coblib,DISP=SHR
//loadlib DD  DSN=idms.loadlib,DISP=SHR
//SYSLMOD DD  DSN=user.loadlib,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSLIN  DD  DSN=&&object,DISP=(OLD,DELETE)
//           DD  *
INCLUDE loadlib(IDMS)           ← Non-CICS only
INCLUDE loadlib(IDMSCINT)       ← CICS only
ENTRY userentry
NAME  userprog(R)
/*
//*

```

### Variable Definitions:

Variable	Definition
accmod	Stepname for batch Command Facility execution of the CREATE ACCESS MODULE statement
compile	Stepname for the compile step
dcmsg	DDname of the system message area (DDLDMSG)
dictb	DDname of the application dictionary definition area (DDLDML)
dictionary-name	Name of the dictionary containing the SQL definition areas
disk	Symbolic device name for workfiles
dloddb	DDname of the application dictionary definition load area (DDLCDOD)
dmcl-name	Name of the DMCL
idms.appldict.ddldcld	Data set name of the application dictionary definition load area (DDLCDOD)

<b>Variable</b>	<b>Definition</b>
idms.appldict.ddldml	Data set name of the application dictionary definition area (DDLDML)
idms.dba.loadlib	Data set name of the load library containing the DMCL and database name table load modules
idms.loadlib	Data set name of the load library containing the CA-IDMS executable modules
idms.sysctl	Data set name of the SYSCTL file
idms.sysmsg.ddldcmsg	Data set name of the system message area (DDLDCMSG)
idms.syssql.ddlcat	Data set name of the SQL definition area (DDLCAT) of the application dictionary
idms.syssql.ddlcatl	Data set name of the SQL definition load area (DDLCATLOD) of the application dictionary
idms.syssql.ddlcatx	Data set name of the SQL definition index area (DDLCATX) of the application dictionary
idms.tapejrn1	Data set name of the tape journal file
loadlib	DDname of the load library containing the CA-IDMS executable modules
precomp	Stepname for the precompile step
sqldd	DDname of the SQL definition area (DDLCAT) of the application dictionary
sqllod	DDname of the SQL definition load area (DDLCATLOD) of the application dictionary
sqlxdd	DDname of the SQL definition index area (DDLCATX) of the application dictionary
sysctl	DDname of the SYSCTL file
sysjrn1	DDname of the tape journal file
sys1.cobol.linklib	Data set name of the library containing the host language compiler module
sys1.coblib	Data set name of the library containing host language compiler subroutines
tape	Symbolic device name for tape journal file
userentry	Entry point for the user program
user.loadlib	Data set name of the load library containing executable modules for user programs
userprog	Name of the user program

<b>Variable</b>	<b>Definition</b>
&&object	Host language compiler output to be passed to the linkage editor
&&source	Precompiler output to be passed to the host language compiler

## A.3 VSE/ESA

**About the examples:** The sample JCL stream that follows contains the steps required to make a host language source program with embedded SQL into form of executable modules. Complete JCL for central version execution is presented, followed by modifications for local mode execution.

The host language for the examples is COBOL. Change the specification of precompiler name, precompiler options, and compiler name according to the host language and version of your program.

Following the sample JCL are a table that gives the meaning of variables used in the examples and a set of usage notes.

**Central version JCL:**

```
*****
**          PRECOMPILE COBOL PROGRAM      **
*****
// EXEC PROC=IDMSLBLs
// DLBL    idmspch,temp.dmlc,0
// EXTENT  SYS020,nnnnnn,,ssss,1111
// ASSGN   SYS020,DISK,VOL=nnnnnn,SHR
// EXEC   IDMSDMLC
Optional precompiler parameters
/*
DMCL=dml-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
Host language source statements with embedded SQL
/*
*****
**          CREATE ACCESS MODULE      **
*****
// EXEC   IDMSBCF
DMCL=dml-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
CREATE ACCESS MODULE statement ;
COMMIT WORK RELEASE ;
/*
*****
**          COMPILE COBOL PROGRAM      **
*****
// DLBL    IJSYSIN,temp.dmlc,0
// EXTENT  SYSIPT,nnnnnn
// ASSGN   SYSIPT,DISK,VOL=nnnnnn,SHR
// OPTION  CATAL,NODECK,NOSYM
// PHASE   userprog,*
// EXEC   FCOBOL
/*
*****
**          LINK PROGRAM MODULE      **
*****
// CLOSE   SYSIPT,SYSRDR
INCLUDE IDMS           ←———— Non-CICS only
INCLUDE IDMScINT       ←———— CICS only
ENTRY(userentry)
// EXEC   LNKEDET
/*

```

**Variable Definitions:**

Variable	Definition
dictionary-name	Name of the dictionary containing the SQL definitions
dml-name	Name of the DMCL
f	File number of the tape journal file
idmspch	Host language compiler output to be passed to the linkage editor
idms.tapejrln	File ID of the tape journal file

Variable	Definition
llll	Number of tracks (CKD) or blocks (FBA) of disk extent
nnnnnn	Volume serial identifier of appropriate disk volume
ssss	Starting track (CKD) or block (FBA) of disk extent
sysjrn1	Filename of the tape journal file
temp.dmlc	File ID of the precompiler output
userentry	Entry point for the user program
userprog	Name of the user program

**Local mode JCL:** To execute in local mode, add these statements to the precompile step:

```
// TLBL    sysjrn1,'idms.tapejrn1',nnnnnn,,f
// ASSGN   SYS009,TAPE,VOL=nnnnnn
```

### A.3.1 Usage

*IDMSLBLS procedure:* IDMSLBLS is a procedure provided during a CA-IDMS VSE/ESA installation. It contains file definitions for these CA-IDMS components:

- Dictionaries
- Demonstration databases
- Disk journal files
- SYSIDMS file

Individual file definitions for these components do not appear in the sample JCL. The IDMSLBLS procedure should be tailored to reflect site-specific names and CA-IDMS VSE/ESA job streams.

*Logical unit assignments:* These logical unit assignments appear in the sample JCL:

- SYS020 — Precompiler output
- SYS009 — Journal file (local mode)

*COBOL internal sort:* For programs that include a COBOL internal sort, place these statements in the compile step, before the EXEC statement:

- ACTION NOAUTO — Prevents multiple inclusions of IDMS
- INCLUDE IDMS — IDMS interface for use with COMRG

- INCLUDE IDMSOPTI — IDMSOPTI module

If IDMSOPTI is included, place this statement after the EXEC PROC=IDMSLBLs statement:

```
// UPSI b
```

where *b* is the appropriate one- through eight-character UPSI switch.

- INCLUDE IDMSCANc — For local mode, abort entry point

## A.4 VM/ESA

**About the examples:** The sample command sequence that follows contains the steps required to make a host language source program with embedded SQL into form of executable modules.

The host language for the example is COBOL. Change the specification of precompiler name, precompiler options, and compiler name according to the host language and version of your program.

Following the example are a table that gives the meaning of variables used in the examples and a set of usage notes.

### Commands for central version execution:

```

/*****+
/*          PRECOMPILE COBOL PROGRAM      */
/*****+
FILEDEF sysipt1 DISK program source a
FILEDEF sysidms1 DISK sysidms1 parms a
FILEDEF syspch DISK prognome COBOL A3
FILEDEF SYSLST PRINTER
OSRUN IDMSDMLC PARM='optional precompiler parameters'

/*****+
/*          CREATE ACCESS MODULE        */
/*****+
FILEDEF sysipt2 DISK create accmod a
FILEDEF sysidms2 DISK sysidms2 parms a
OSRUN IDMSBCF

/*****+
/*          COMPILE COBOL PROGRAM      */
/*****+
FILEDEF TEXT DISK prognome TEXT A3
COBOL prognome (OSDECK APOST LIB
TXTLIB DEL utextlib prognome
TXTLIB ADD utextlib prognome

/*****+
/*          LINK PROGRAM MODULE        */
/*****+
FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD uoloadlib LOADLIB A6 (RECFM V LRECL 1024 BLKSIZE 1024
FILEDEF objlib DISK utextlib TXTLIB a
FILEDEF SYSLIB DISK coblibvs TXTLIB p
LKED linkctl (LIST XREF LET MAP RENT NOTERM PRINT SIZE 512K 64K

```

Linkage editor control statements (in *linkctl*):

```

INCLUDE objlib(prognome)
INCLUDE objlib1(IDMS)
ENTRY prognome
NAME   prognome(R)

```

### Variable Definitions:

Variable	Definition
coblibvs <i>TEXTLIB p</i>	Filename, filetype, and filemode of the library that contains host language compiler modules
create accmod a	Filename of the file containing the CREATE ACCESS MODULE statement
linkctl	Filename of the file that contains the linkage editor control statements

Variable	Definition
loadlib	DDname of the load library containing the CA-IDMS executable modules
objlib	DDname of the user object library
objlib1	DDname of the CA-IDMS object library
program <i>COBOL A3</i>	Filename, filetype, and filemode of the precompiler output
progname	Name of the user program
program source a	Filename of the file containing the program source
sysidms1	DDname for the file of SYSIDMS parameters for the precompiler step
sysidms1 parms a	Filename of the file containing SYSIDMS parameters for the precompiler step
sysidms2	DDname for the file of SYSIDMS parameters for the step to create the access module
sysidms2 parms a	Filename of the file containing SYSIDMS parameters for the step to create the access module
sysipt1	DDname for the program source file
sysipt2	DDname for the file containing the CREATE ACCESS MODULE statement
syspch	DDname for the precompiler output
uloadlib <i>LOADLIB A6</i>	Filename, filetype, and filemode of the user load library
utextlib <i>TXTLIB a</i>	Filename, filetype, and filemode of the user text library

### A.4.1 Usage

**Local mode:** To specify that the precompiler is executing in local mode, perform one of the following:

- Link the program with an IDMSOPTI program that specifies local execution mode
- Specify \*LOCAL\* as the first input parameter of the filename, type and mode identified by idmspass input a in the IDMSFD exec.
- Modify the OSRUN statement:

```
OSRUN IDMSDMCL PARM='*LOCAL*'
```

**Note:** This option is valid only if the OSRUN command is issued from a System Product interpreter or an EXEC2 file.

---

A local mode job should contain file definitions to include the following in the precompile step and the step to create the access module:

**Variable Definitions:**

Variable	Definition
dcmsg	DDname of the system message area (DDLDCMSG)
dictb	DDname of the application dictionary definition area (DDLDML)
dloddb	DDname of the application dictionary definition load area (DDLCDLOD)
idms.appldict.ddldclod	File name of the application dictionary definition load area (DDLCDLOD)
idms.appldict.ddldml	File name of the application dictionary definition area (DDLDML)
idms.sysmsg.ddldcmsg	File name of the system message area (DDLDCMSG)
idms.syssql.ddlcat	File name of the SQL definition area (DDLCAT) of the application dictionary
idms.syssql.ddlcatl	File name of the SQL definition load area (DDLCATLOD) of the application dictionary
idms.syssql.ddlcatx	File name of the SQL definition index area (DDLCATX) of the application dictionary
idms.tapejrn1	File name of the tape journal file
sqldd	DDname of the SQL definition area (DDLCAT) of the application dictionary
sqllod	DDname of the SQL definition load area (DDLCATLOD) of the application dictionary
sqlxdd	DDname of the SQL definition index area (DDLCATX) of the application dictionary
sysjrn1	DDname of the tape journal file

A local mode job should contain file definitions to include the following in the step to create the access module:

**SYSIPT file:** To create a sysipt file:

1. On the VM/ESA command line, type **XEDIT sysipt data a (NOPROF** and press [Enter]
2. On the XEDIT command line, type **INPUT** and press [Enter]
3. In input mode, type in the IDMSPASS input parameters
4. Press [Enter] to exit input mode
5. On the XEDIT command line, type **FILE** and press [Enter]

**SYSIDMS file:** To execute the precompiler and create the access module, you should include these SYSIDMS parameters:

- DMCL=*dmcl-name*, to identify the DMCL
- DICTNAME=*dictionary-name*, to identify the dictionary whose catalog component contains the database definitions

To create a file of SYSIDMS parameters:

1. On the VM/ESA command line, type **XEDIT sysidms data a (NOPROF** and press [Enter]
2. On the XEDIT command line, type **INPUT** and press [Enter]
3. In input mode, type in the SYSIDMS parameters
4. Press [Enter] to exit input mode
5. On the XEDIT command line, type **FILE** and press [Enter]

**Note:** Refer to *CA-IDMS Database Administration* for documentation of SYSIDMS parameters.

## A.5 BS2000/OSD

**About the examples:** The sample JCL stream that follows contains the steps required to make a host language source program with embedded SQL into form of executable modules.

The host language in the example is COBOL 85. Change the specification of precompiler name, precompiler options, and compiler name according to the host language and version of your program.

Following the example is a table that gives the meaning of variables used in the examples.

### Central Version JCL

```

/REMARK ****
/REMARK * Establish environment *
/REMARK ****
/ADD-FILE-LINK L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSLDR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=*DUMMY
/ASSIGN-SYSLST TO=syslst-name
/REMARK ****
/REMARK * Precompile Cobol program *
/REMARK ****
/CRE-FILE F-NAME=temp.sys001, SUPPRESS-ERRORS=*FILE-EXISTING,
/   SUP=PUB-DISK(SPACE=RELA(PRIM-ALLOC=60,SEC-ALLOC=30)) -
/ADD-FILE-LINK L-NAME=SYS001,F-NAME=temp.sys001
/CRE-FILE F-NAME=temp.sys002, SUPPRESS-ERRORS=*FILE-EXISTING,
/   SUP=PUB-DISK(SPACE=RELA(PRIM-ALLOC=60,SEC-ALLOC=30)) -
/ADD-FILE-LINK L-NAME=SYS002,F-NAME=temp.sys002
/CRE-FILE F-NAME=temp.sys003, SUPPRESS-ERRORS=*FILE-EXISTING,
/   SUP=PUB-DISK(SPACE=RELA(PRIM-ALLOC=60,SEC-ALLOC=30)) -
/ADD-FILE-LINK L-NAME=SYS003,F-NAME=temp.sys003
/ASSIGN-SYSOPT TO=cobol-input
/ASSIGN-SYSDTA TO=program-source
/START-PROG *MOD(ELEM=IDMSDMLC,LIB=idms.dba.loadlib,RUN-MODE=*ADV)
DMCL=dmcl-name
DICTNAME=dictionary-name
PARM='optional precompiler parameters'
Additional SYSIDMS parameters as appropriate
END-SYSIDMS
Host language source statements with embedded SQL
/ASSIGN-SYSOPT TO=*PRIMARY

```

```

/REMARK ****
/REMARK * Create access module *
/REMARK ****
/ASSIGN-SYSDTA TO=SYSDTA
/START-PROG *MOD(ELEM=IDMSBCF,LIB=idms.loadlib,RUN-MODE=*ADV)
DMCL=dmcl-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters as appropriate
END-SYSIDMS
CREATE ACCESS MODULE statement;
COMMIT WORK RELEASE ;
/REMARK ****
/REMARK * Compile the program *
/REMARK ****
/DELETE-SYSTEM-FILE SYSTEM-FILE=*OMF
/ASSIGN-SYSDTA TO=cobol-input
/START-COBOL85-COMPILER
/   MODULE-OUTPUT=*OMF,
/   COMPILER-ACTION=MODULE-GENERATION(MODULE-FORMAT=OM),
/   LISTING=(SOURCE=YES,DIAGNOSTICS=YES,OUTPUT=SYSLST)
/REMARK ****
/REMARK * Link the program *
/REMARK ****
/START-BINDER
//START-LLM-CREATION INTERNAL-NAME=userprog
//INC-MOD LIB=*OMF
//INC-MOD LIB=idms.loadlib,ELEM=IDMSPBS2
//RESOLVE-BY-AUTOLINK LIB=cobol85.resovelib
//SAVE-LLM LIB=idms.loadlib.user,ELEM=userprog(VER=@),OVER=YES
//END

```

**Local Mode JCL:** To compile the program in local mode, replace the ADD-FILE-LINK statement for the sysctl file with these statements.

```

/ADD-FILE-LINK L-NAME=J1JRNL,F-NAME=*DUMMY
/ADD-FILE-LINK L-NAME=J2JRNL,F-NAME=*DUMMY
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.ddldml,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.ddldcld,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sqldd,F-NAME=idms.syssql.ddlcat,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sqlxdd,F-NAME=idms.syssql.ddlcatx,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sqllod,F-NAME=idms.syssql.ddlld,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmmsg,F-NAME=idms.sysmsg.ddldcmsg,SHARED-UPD=*YES

```

### Variable Definitions:

Variable	Definition
cobol-input	Host language compiler output to be passed to the linkage editor
cobol85.resovelib	Filename of the COBOL runtime object library
dcmmsg	Linkname of the system message area (DDLDCMSG)
dictb	Linkname of the application dictionary definition area (DDLDML)

<b>Variable</b>	<b>Definition</b>
dictionary-name	Name of the dictionary containing the SQL definition areas
dloddb	Linkname of the application dictionary definition load area (DDLDCLOD)
dmcl-name	Name of the DMCL
idms.appldict.ddldclod	Filename of the application dictionary definition load area (DDLDCLOD)
idms.appldict.ddldml	Filename of the application dictionary definition area (DDLDML)
idms.dba.loadlib	Filename of the load library containing the DMCL and database name table load modules
idms.loadlib	Filename of the load library containing
idms.loadlib.user	Filename of the load library containing the user-defined executable modules
idms.sysctl	Filename of the SYSCTL file
idms.sysmsg.ddldcmsg	Filename of the system message area (DDLDMSG)
idms.syssql.ddlcat	Filename of the SQL definition area (DDLCAT)
idms.syssql.ddlcatl	Filename of the SQL definition load area (DDLCATLOD)
idms.syssql.ddlcatx	Filename of the SQL definition index area (DDLCATX)
sqldd	Linkname of the SQL definition area (DDLCAT)
sqllod	Linkname of the SQL definition load area (DDLCATLOD)
sqlxdd	Linkname of the SQL definition index area (DDLCATX)
sysctl	Linkname of the SYSCTL file
temp.sys001 temp.sys002 temp.sys003	Filenames of temporary work files
userprog	Name of the user program



## Appendix B. Test Database

---

B.1	About this appendix	B-3
B.2	Table names and descriptions	B-4
B.2.1	ASSIGNMENT	B-4
B.2.2	BENEFITS	B-5
B.2.3	CONSULTANT	B-6
B.2.4	COVERAGE	B-6
B.2.5	DEPARTMENT	B-6
B.2.6	DIVISION	B-7
B.2.7	EMPLOYEE	B-7
B.2.8	EXPERTISE	B-7
B.2.9	INSURANCE_PLAN	B-8
B.2.10	JOB	B-8
B.2.11	POSITION	B-9
B.2.12	PROJECT	B-9
B.2.13	SKILL	B-9
B.3	Test data	B-10
B.3.1	Departments	B-10
B.3.2	Divisions	B-10
B.3.3	Insurance plans	B-11
B.3.4	Jobs	B-11
B.3.5	Projects	B-12
B.3.6	Skills	B-12
B.4	Test database DDL	B-14
B.5	Demo Data	B-22



## B.1 About this appendix

This appendix provides complete information about the data in the test database, supplied with CA-IDMS, to which most of the sample programs in this manual refer. You can use this information to develop SQL programs that access the test database.

## B.2 Table names and descriptions

### B.2.1 ASSIGNMENT

EMP_ID	Employee ID
PROJ_ID	ID of project to which employee is assigned
START_DATE	Date employee was assigned to the project
END_DATE	Date employee completed work on the project

## B.2.2 BENEFITS

FISCAL_YEAR	Fiscal year for which this data applies
EMP_ID	Employee ID
VAC_ACCRUED	Vacation <b>hours</b> accrued to date
VAC_TAKEN	Vacation <b>hours</b> taken to date
SICK_ACCRUED	Sick <b>days</b> accrued to date
SICK_TAKEN	Sick <b>days</b> taken to date
STOCK_PERCENT	Percentage of earnings allocated to stock purchase
STOCK_AMOUNT	Year-to-date amount deducted for stock purchase
LAST REVIEW DATE	Date of last employee review
REVIEW_PERCENT	Percent increase at last review
PROMO_DATE	Date of last promotion
RETIRE_PLAN	Retirement fund identifier: STOCK, BONDS, 401K
RETIRE_PERCENT	Percentage of earnings deducted for retirement
BONUS_AMOUNT	Amount of last bonus
COMP_ACCRUED	<b>Hours</b> of compensation time accrued
COMP_TAKEN	<b>Hours</b> of compensation time taken
EDUC_LEVEL	Level of education: GED, HSDIP, JRCOLL, COLL, MAS, PHD
UNION_ID	Union identification number
UNION_DUES	Amount of dues deducted <b>per pay period</b>

### B.2.3 CONSULTANT

CON_ID	Unique consultant ID
CON_FNAME	Consultant's first name
CON_LNAME	Consultant's last name
MANAGER_ID	Employee ID of consultant's manager
DEPT_ID	ID of department to which consultant is assigned
PROJ_ID	ID of project to which consultant is assigned
STREET	Consultant's street address
CITY	Consultant's city
STATE	Consultant's state
ZIP_CODE	Consultant's zip code
PHONE	Consultant's phone
BIRTH_DATE	Birth date
START_DATE	Consultant's date of hire
SS_NUMBER	Social security number
RATE	Hourly rate of pay

### B.2.4 COVERAGE

PLAN_CODE	Code of insurance plan providing the coverage
EMP_ID	Employee ID of employee having the coverage
SELECTION_DATE	Date employee selected this insurance plan
TERMINATION_DATE	Date employee terminated this insurance plan; if null, plan is still in force
NUM_DEPENDENTS	Number of <b>dependents</b> covered under this insurance plan

### B.2.5 DEPARTMENT

DEPT_ID	Unique department ID
DEPT_HEAD_ID	Employee ID of department head
DIV_CODE	Code of the division to which this department belongs
DEPT_NAME	Department name

## B.2.6 DIVISION

DIV_CODE	Unique division ID
DIV_HEAD_ID	Employee ID of division head
DIV_NAME	Division name

## B.2.7 EMPLOYEE

EMP_ID	Unique employee ID
MANAGER_ID	Employee ID of employee's manager
EMP_FNAME	Employee's first name
EMP_LNAME	Employee's last name
DEPT_ID	ID of department to which employee is assigned
STREET	Employee's street address
CITY	Employee's city
STATE	Employee's state
ZIP_CODE	Employee's zip code
PHONE	Employee's phone
STATUS	Status of employee: (A) Active; (S) Short-term disability; (L) Long term disability
SS_NUMBER	Social security number
START_DATE	Employee's date of hire
TERMINATION_DATE	Date of termination
BIRTH_DATE	Birth date

## B.2.8 EXPERTISE

EMP_ID	Employee ID
SKILL_ID	Skill ID
SKILL_LEVEL	Level of ability in this skill: 01 (low) to 04 (high)
EXP_DATE	Date this level of ability was achieved

### B.2.9 INSURANCE\_PLAN

PLAN_CODE	Unique plan code for company offering the insurance
COMP_NAME	Name of insurance company
STREET	Street address of insurance company
CITY	City address of insurance company
STATE	State address of insurance company
ZIP_CODE	Zip code of insurance company
PHONE	Telephone number of insurance company
GROUP_NUMBER	Commonwealth's group number for this insurance company
DEDUCT	Dollar amount deductible <b>per year</b> for this insurance plan
MAX_LIFE_BENEFIT	Maximum dollar amount to be paid to insured employee
FAMILY_COST	Amount deducted <b>per paycheck</b> for family coverage
DEP_COST	Additional amount deducted per paycheck per dependent
EFF_DATE	Date this coverage plan became effective

### B.2.10 JOB

JOB_ID	Unique job ID
JOB_TITLE	Job title
MIN_RATE	Minimum salary/hourly rate for this job
MAX_RATE	Maximum salary/hourly rate for this job
SALARY_IND	Indicator for type of salary: (S) salaried; (H) hourly
NUM_OF_POSITIONS	Total number of positions for this job
NUM_OPEN	Number of positions currently open
EFF_DATE	Date this job became effective
JOB_DESLINE_1	First line of job description
JOB_DESLINE_2	Second line of job description

### B.2.11 POSITION

EMP_ID	Employee ID
JOB_ID	Job ID associated with this employee
START_DATE	Date employee began this job
FINISH_DATE	Date employee ended this job (null if current)
HOURLY_RATE	Hourly rate earned while in this job (if hourly position)
SALARY_AMOUNT	Yearly salary earned while in this job (if salaried position)
BONUS_PERCENT	Bonus percent amount for this position (if sales position)
COMM_PERCENT	Commission percent for this position (if sales position)
OVERTIME_RATE	Overtime rate for this position (if hourly position)

### B.2.12 PROJECT

PROJ_ID	Unique project ID
PROJ_LEADER_ID	Employee ID of project leader
EST_START_DATE	Estimated date project is to begin
EST_END_DATE	Estimated date project is to end
ACT_START_DATE	Actual date project began
ACT_END_DATE	Actual date project ended
EST_MAN_HOURS	Total number of hours estimated for project
ACT_MAN_HOURS	Actual number of hours required for project
PROJ_DESC	Project description

### B.2.13 SKILL

SKILL_ID	Unique skill ID
SKILL_NAME	Skill name
SKILL_DESC	Skill description

## B.3 Test data

This section lists the test data stored in the test database.

### B.3.1 Departments

Code	Name	Division code	Head ID
3510	Appraisal - Used cars	D02	3082
2200	Sales - Used cars	D02	2180
1100	Purchasing - Used cars	D02	2246
3520	Appraisal - New cars	D04	3769
2210	Sales - New cars	D04	2010
4200	Leasing - New cars	D04	1003
1110	Purchasing - New cars	D04	1765
1120	Purchasing - Service	D06	2004
4600	Maintenance	D06	2096
3530	Appraisal - Service	D06	2209
5100	Billing	D06	2598
6200	Corporate Administration	D09	2461
5200	Corporate Marketing	D09	2894
5000	Corporate Accounting	D09	2466
4900	MIS	D09	2466
6000	Legal	D09	1003
4500	Human Resources	D09	3222

### B.3.2 Divisions

Division code	Division name	Head ID
D02	Used cars	2180
D04	New cars	2010
D06	Service	4321
D09	Corporate	1003

### B.3.3 Insurance plans

Plan ID	Name
PLI	Providential Life Insurance
HHM	Homostasis Health Maintenance Program
HGH	Holistic Group Health Association
DAS	Dental Associates

### B.3.4 Jobs

Job ID	Name	Minimum salary	Maximum salary	Salaried/ hourly	No.
8001	Vice president	90000	136000	S	1
4023	Accountant	44000	120000	S	1
2051	AP Clerk	8.80	14.60	H	2
2053	AR Clerk	8.80	14.60	H	3
2077	Purch Clerk	17000	30000	S	3
3029	Computer Operator	25500	44000	S	1
3051	Data Entry Clerk	8.50	11.45	H	1
6011	Manager - Acctng	59400	121000	S	1
4560	Mechanic	11.45	21.00	H	7
4666	Sr Mechanic	41000	91000	S	1
4734	Mktng Admin	25000	62000	S	2
3333	Sales Trainee	21600	39000	S	4
5555	Salesperson	30000	79500	S	9
6004	Manager - HR	66000	138000	S	1
6021	Manager - Mktng	76000	150000	S	1
2055	PAYROLL CLERK	17000	30000	S	1
4025	Writer - Mktng	31000	50000	S	1
9001	President	111000	190000	S	1
4123	Recruiter	35000	56000	S	1
4130	Benefits Analyst	35000	56000	S	1
4012	Admin Asst	21000	44000	S	4

### B.3 Test data

---

Job ID	Name	Minimum salary	Maximum salary	Salaried/ hourly	No.
5111	CUST SER REP	27000	54000	S	4
4700	Purch Agent	33000	60000	S	5
5890	Appraisal Spec	45000	70000	S	5
5110	CUST SERVICE MGR	40000	108000	S	1

### B.3.5 Projects

Project ID	Description
P634	TV ads - WTVK
C200	New brand research
P400	Christmas media
C203	Consumer study
C240	Service study
D880	System analysis

### B.3.6 Skills

Skill ID	Name
4444	Assembly
3333	Bodywork
3088	Brake work
3065	Electronics
1030	Acct Mgt
5130	Basic math
5160	Calculus
4250	Data entry
4370	Filing
5200	General Acctng
5500	General Mktng
5430	Mktng Writing
5420	Writing

<b>Skill ID</b>	<b>Name</b>
4490	Gen Ledger
4430	Interviewing
1000	Management
4420	Telephone
5180	Statistics
4410	Typing
5309	Appraising
6770	Purchasing
7000	Sales
6666	Billing
6650	Diesel Engine Repair
6670	Gas Engine Repair
6470	Window Installation

## B.4 Test database DDL

This appendix contains the SQL DDL that creates the demonstration database provided with the installation of CA-IDMS.

```
*****
* Create schema for the following tables. Then set session qualifier
* for that schema
*****
CREATE SCHEMA DEMOEMPL;
SET SESSION CURRENT SCHEMA DEMOEMPL;
*****
* Create the tables that belong to the schema DEMOEMPL. Each
* table is associated with an area in the segment DEMOEMPL.
*****
CREATE TABLE BENEFITS
(FISCAL_YEAR      UNSIGNED NUMERIC(4,0)      NOT NULL,
 EMP_ID           UNSIGNED NUMERIC(4,0)      NOT NULL,
 VAC_ACCRUED      UNSIGNED DECIMAL(6,2)     NOT NULL WITH DEFAULT,
 VAC_TAKEN        UNSIGNED DECIMAL(6,2)     NOT NULL WITH DEFAULT,
 SICK_ACCRUED    UNSIGNED DECIMAL(6,2)     NOT NULL WITH DEFAULT,
 SICK_TAKEN       UNSIGNED DECIMAL(6,2)     NOT NULL WITH DEFAULT,
 STOCK_PERCENT   UNSIGNED DECIMAL(6,3)      NOT NULL WITH DEFAULT,
 STOCK_AMOUNT     UNSIGNED DECIMAL(10,2)     NOT NULL WITH DEFAULT,
 LAST_REVIEW_DATE DATE                      ,
 REVIEW_PERCENT  UNSIGNED DECIMAL(6,3)      ,
 PROMO_DATE       DATE                      ,
 RETIRE_PLAN     CHAR(6)                   ,
 RETIRE_PERCENT  UNSIGNED DECIMAL(6,3)      ,
 BONUS_AMOUNT    UNSIGNED DECIMAL(10,2)     ,
 COMP_ACCRUED    UNSIGNED DECIMAL(6,2)      NOT NULL WITH DEFAULT,
 COMP_TAKEN      UNSIGNED DECIMAL(6,2)      NOT NULL WITH DEFAULT,
 EDUC_LEVEL      CHAR(06)                   ,
 UNION_ID         CHAR(10)                   ,
 UNION_DUES       UNSIGNED DECIMAL(10,2)     ,
 CHECK ( (RETIRE_PLAN IN ('STOCK', 'BONDS', '401K') ) AND
          (EDUC_LEVEL IN ('GED', 'HSDIP', 'JRCOLL', 'COLL',
                           'MAS', 'PHD') ) )
 IN SQLDEMO.EMPLAREA;

*****
CREATE TABLE COVERAGE
(PLAN_CODE        CHAR(03)                   NOT NULL,
 EMP_ID           UNSIGNED NUMERIC(4,0)      NOT NULL,
 SELECTION_DATE   DATE                      NOT NULL WITH DEFAULT,
 TERMINATION_DATE DATE                      ,
 NUM_DEPENDENTS  UNSIGNED NUMERIC(2,0)      NOT NULL WITH DEFAULT)
 IN SQLDEMO.EMPLAREA;

*****
CREATE TABLE DEPARTMENT
(DEPT_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 DEPT_HEAD_ID    UNSIGNED NUMERIC(4,0)      ,
 DIV_CODE         CHAR(03)                   NOT NULL,
 DEPT_NAME        CHAR(40)                   NOT NULL)
 IN SQLDEMO.INFOAREA;
```

```
*****
CREATE TABLE      DIVISION
(DIV_CODE        CHAR(03)          NOT NULL,
 DIV_HEAD_ID     UNSIGNED NUMERIC(4,0) ,
 DIV_NAME        CHAR(40)          NOT NULL)
IN SQLDEMO.INFOAREA;

*****
CREATE TABLE      EMPLOYEE
(EMP_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 MANAGER_ID      UNSIGNED NUMERIC(4,0)      ,
 EMP_FNAME       CHAR(20)           NOT NULL,
 EMP_LNAME       CHAR(20)           NOT NULL,
 DEPT_ID         UNSIGNED NUMERIC(4,0)      NOT NULL,
 STREET          CHAR(40)           NOT NULL,
 CITY            CHAR(20)           NOT NULL,
 STATE           CHAR(02)           NOT NULL,
 ZIP_CODE        CHAR(09)           NOT NULL,
 PHONE           CHAR(10)           ,
 STATUS          CHAR               NOT NULL,
 SS_NUMBER       UNSIGNED NUMERIC(9,0)      NOT NULL,
 START_DATE      DATE               NOT NULL,
 TERMINATION_DATE DATE               ,
 BIRTH_DATE      DATE               ,
 CHECK` ( (EMP_ID <= 8999 ) AND (STATUS IN ('A', 'S', 'L', 'T') ) ) )
IN SQLDEMO.EMPLAREA;

*****
CREATE TABLE      INSURANCE_PLAN
(PLAN_CODE       CHAR(03)          NOT NULL,
 COMP_NAME       CHAR(40)          NOT NULL,
 STREET          CHAR(40)           NOT NULL,
 CITY            CHAR(20)           NOT NULL,
 STATE           CHAR(02)           NOT NULL,
 ZIP_CODE        CHAR(09)           NOT NULL,
 PHONE           CHAR(10)           NOT NULL,
 GROUP_NUMBER    UNSIGNED NUMERIC(4,0)      NOT NULL,
 DEDUCT          UNSIGNED DECIMAL(9,2)      ,
 MAX_LIFE_BENEFIT UNSIGNED DECIMAL(9,2)      ,
 FAMILY_COST     UNSIGNED DECIMAL(9,2)      ,
 DEP_COST         UNSIGNED DECIMAL(9,2)      ,
 EFF_DATE        DATE               NOT NULL)
IN SQLDEMO.INFOAREA;
```

#### B.4 Test database DDL

---

```
CREATE TABLE      JOB
(JOB_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 JOB_TITLE       CHAR(20)                      NOT NULL,
 MIN_RATE        UNSIGNED DECIMAL(10,2)        ,
 MAX_RATE        UNSIGNED DECIMAL(10,2)        ,
 SALARY_IND      CHAR(01)                      ,
 NUM_OF_POSITIONS UNSIGNED DECIMAL(4,0)        ,
 EFF_DATE        DATE                          ,
 JOB_DESC_LINE_1 VARCHAR(60)                   ,
 JOB_DESC_LINE_2 VARCHAR(60)                   ,
 CHECK ( SALARY_IND IN ('S', 'H') ) )
IN SQLDEMO.INFOAREA;
*****
CREATE TABLE      POSITION
(EMP_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 JOB_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 START_DATE      DATE                         NOT NULL,
 FINISH_DATE     DATE                         ,
 HOURLY_RATE     UNSIGNED DECIMAL(7,2)        ,
 SALARY_AMOUNT    UNSIGNED DECIMAL(10,2)        ,
 BONUS_PERCENT   UNSIGNED DECIMAL(7,3)        ,
 COMM_PERCENT    UNSIGNED DECIMAL(7,3)        ,
 OVERTIME_RATE   UNSIGNED DECIMAL(5,2)        ,
 CHECK ( (HOURLY_RATE IS NOT NULL AND SALARY_AMOUNT IS NULL)
        OR (HOURLY_RATE IS NULL AND SALARY_AMOUNT IS NOT NULL) ) )
IN SQLDEMO.EMPLAREA;
*****
```

```
*****
CREATE SCHEMA DEMOPROJ;
SET SESSION CURRENT SCHEMA DEMOPROJ;

*****
* Create the tables that belong to the schema DEMOPROJ. Each
* table is associated with an area in the segment PROJSEG.
*****



CREATE TABLE      ASSIGNMENT
(EMP_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 PROJ_ID          CHAR(10)                     NOT NULL,
 START_DATE       DATE                         NOT NULL,
 END_DATE         DATE                         )
IN PROJSEG.PROJAREA;

*****



CREATE TABLE      CONSULTANT
(CON_ID           UNSIGNED NUMERIC(4,0)      NOT NULL,
 CON_FNAME        CHAR(20)                     NOT NULL,
 CON_LNAME        CHAR(20)                     NOT NULL,
 MANAGER_ID       UNSIGNED NUMERIC(4,0)      NOT NULL,
 DEPT_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 PROJ_ID          CHAR(10)                     ,
 STREET           CHAR(40)                     ,
 CITY             CHAR(20)                     NOT NULL,
 STATE            CHAR(02)                     NOT NULL,
 ZIP_CODE         CHAR(09)                     NOT NULL,
 PHONE            CHAR(10)                     ,
 BIRTH_DATE       DATE                         ,
 START_DATE       DATE                         NOT NULL,
 SS_NUMBER        UNSIGNED NUMERIC(9,0)      NOT NULL,
 RATE             UNSIGNED DECIMAL(7,2)      ,
 CHECK ( (CON_ID >= 9000 AND CON_ID <= 9999) ) )
IN PROJSEG.PROJAREA;

*****



CREATE TABLE      EXPERTISE
(EMP_ID          UNSIGNED NUMERIC(4,0)      NOT NULL,
 SKILL_ID         UNSIGNED NUMERIC(4,0)      NOT NULL,
 SKILL_LEVEL     CHAR(02)                     ,
 EXP_DATE         DATE                         )
IN PROJSEG.PROJAREA;

*****
```

```
*****
CREATE TABLE PROJECT
(PROJ_ID CHAR(10) NOT NULL,
PROJ_LEADER_ID UNSIGNED NUMERIC(4,0) ,
EST_START_DATE DATE ,
EST_END_DATE DATE ,
ACT_START_DATE DATE ,
ACT_END_DATE DATE ,
EST_MAN_HOURS UNSIGNED DECIMAL(7,2) ,
ACT_MAN_HOURS UNSIGNED DECIMAL(7,2) ,
PROJ_DESC VARCHAR(60) NOT NULL)
IN PROJSEG.PROJAREA;
*****



CREATE TABLE SKILL
(SKILL_ID UNSIGNED NUMERIC(4,0) NOT NULL,
SKILL_NAME CHAR(20) NOT NULL,
SKILL_DESC VARCHAR(60) )
IN PROJSEG.PROJAREA;
*****



* Name calc keys for above tables (in order that they were defined)
*****



CREATE UNIQUE CALC KEY ON DEMOEMPL.DEPARTMENT(DEPT_ID);

CREATE UNIQUE CALC KEY ON DEMOEMPL.DIVISION(DIV_CODE);

CREATE UNIQUE CALC KEY ON DEMOEMPL.EMPLOYEE(EMP_ID);

CREATE UNIQUE CALC KEY ON DEMOEMPL.INSURANCE_PLAN(PLAN_CODE);

CREATE UNIQUE CALC KEY ON DEMOEMPL.JOB(JOB_ID);

CREATE UNIQUE CALC KEY ON DEMOPROJ.CONSULTANT(CON_ID);

CREATE UNIQUE CALC KEY ON DEMOPROJ.PROJECT(PROJ_ID);

CREATE UNIQUE CALC KEY ON DEMOPROJ.SKILL(SKILL_ID);

*****



* Create unique indexes for tables in order in which they were defined
*****



CREATE UNIQUE INDEX AS_EMPPROJ_NDX ON
DEMOPROJ.ASSIGNMENT(EMP_ID,PROJ_ID);

CREATE UNIQUE INDEX EX_EMPSKILL_NDX ON
DEMOPROJ.EXPERTISE(EMP_ID, SKILL_ID);
```

```
*****
* Create nonunique indexes for tables in order in which they
* were defined
*****
CREATE INDEX CO_CODE_NDX ON DEMOEMPL.COVERAGE(PLAN_CODE)
IN SQLDEMO.INDXAREA;

CREATE INDEX DE_CODE_NDX ON DEMOEMPL.DEPARTMENT(DIV_CODE);

CREATE INDEX DI_HEAD_NDX ON DEMOEMPL.DIVISION(DIV_HEAD_ID);

CREATE INDEX DE_HEAD_NDX ON DEMOEMPL.DEPARTMENT(DEPT_HEAD_ID);

CREATE INDEX EM_MANAGER_NDX ON DEMOEMPL.EMPLOYEE(MANAGER_ID)
IN SQLDEMO.INDXAREA;

CREATE INDEX EM_NAME_NDX ON DEMOEMPL.EMPLOYEE(EMP_LNAME, EMP_FNAME)
IN SQLDEMO.INDXAREA;

CREATE INDEX EM_DEPT_NDX ON DEMOEMPL.EMPLOYEE(DEPT_ID)
IN SQLDEMO.INDXAREA;

CREATE INDEX IN_NAME_NDX ON DEMOEMPL.INSURANCE_PLAN(COMP_NAME)
COMPRESSED;

CREATE INDEX PO_JOB_NDX ON DEMOEMPL.POSITION(JOB_ID)
IN SQLDEMO.INDXAREA;

CREATE INDEX CN_NAME_NDX
ON DEMOPROJ.CONSULTANT(CON_LNAME,CON_FNAME);

*****
* Create referential constraints
*****
CREATE CONSTRAINT EMP_BENEFITS
DEMOEMPL.BENEFITS (EMP_ID) REFERENCES
DEMOEMPL.EMPLOYEE (EMP_ID)
LINKED CLUSTERED
ORDER BY (FISCAL_YEAR DESC);

CREATE CONSTRAINT INSPLAN_COVERAGE
DEMOEMPL.COVERAGE (PLAN_CODE) REFERENCES
DEMOEMPL.INSURANCE_PLAN (PLAN_CODE)
UNLINKED;

CREATE CONSTRAINT EMP_COVERAGE
DEMOEMPL.COVERAGE (EMP_ID) REFERENCES
DEMOEMPL.EMPLOYEE (EMP_ID)
LINKED CLUSTERED
ORDER BY (PLAN_CODE) UNIQUE;
```

```
CREATE CONSTRAINT DIVISION_DEPT
    DEMOEMPL.DEPARTMENT (DIV_CODE) REFERENCES
    DEMOEMPL.DIVISION (DIV_CODE)
    UNLINKED;

CREATE CONSTRAINT EMP_DEPT_HEAD
    DEMOEMPL.DEPARTMENT (DEPT_HEAD_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    UNLINKED;

CREATE CONSTRAINT EMP_DIV_HEAD
    DEMOEMPL.DIVISION (DIV_HEAD_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    UNLINKED;

CREATE CONSTRAINT DEPT_EMPLOYEE
    DEMOEMPL.EMPLOYEE (DEPT_ID) REFERENCES
    DEMOEMPL.DEPARTMENT (DEPT_ID)
    UNLINKED;

CREATE CONSTRAINT MANAGER_EMP
    DEMOEMPL.EMPLOYEE (MANAGER_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    UNLINKED;

CREATE CONSTRAINT SKILL_EXPERTISE
    DEMOPROJ.EXPERTISE (SKILL_ID) REFERENCES
    DEMOPROJ.SKILL (SKILL_ID)
    LINKED CLUSTERED;

CREATE CONSTRAINT EMP_POSITION
    DEMOEMPL.POSITION (EMP_ID) REFERENCES
    DEMOEMPL.EMPLOYEE (EMP_ID)
    LINKED CLUSTERED
    ORDER BY (JOB_ID) UNIQUE;

CREATE CONSTRAINT JOB_POSITION
    DEMOEMPL.POSITION (JOB_ID) REFERENCES
    DEMOEMPL.JOB (JOB_ID)
    UNLINKED;

CREATE CONSTRAINT PROJECT_ASSIGN
    DEMOPROJ.ASSIGNMENT (PROJ_ID) REFERENCES
    DEMOPROJ.PROJECT (PROJ_ID)
    LINKED CLUSTERED;

CREATE CONSTRAINT PROJECT_CONSULT
    DEMOPROJ.CONSULTANT (PROJ_ID) REFERENCES
    DEMOPROJ.PROJECT (PROJ_ID)
    LINKED INDEX
    ORDER BY (PROJ_ID);
```

```
*****
* Alter tables to remove default indexes as necessary
*****
ALTER TABLE DEMOEMPL.COVERAGE
    DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.DEPARTMENT
    DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.DIVISION
    DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.EMPLOYEE
    DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.INSURANCE_PLAN
    DROP DEFAULT INDEX;

ALTER TABLE DEMOEMPL.POSITION
    DROP DEFAULT INDEX;

ALTER TABLE DEMOPROJ.ASSIGNMENT
    DROP DEFAULT INDEX;

ALTER TABLE DEMOPROJ.CONULTANT
    DROP DEFAULT INDEX;

ALTER TABLE DEMOPROJ.EXPERTISE
    DROP DEFAULT INDEX;

*****
* Create views
*****
CREATE VIEW DEMOEMPL.EMP_VACATION
    (EMP_ID, DEPT_ID, VAC_TIME)
AS SELECT E.EMP_ID, DEPT_ID, SUM(VAC_ACCRUED) - SUM(VAC_TAKEN)
    FROM DEMOEMPL.EMPLOYEE E, DEMOEMPL.BENEFITS B
    WHERE E.EMP_ID = B.EMP_ID
    GROUP BY DEPT_ID, E.EMP_ID;

CREATE VIEW DEMOEMPL.OPEN_POSITIONS
    (JOB_ID, JOB_NAME, OPEN_POS)
AS SELECT J.JOB_ID, J.JOB_TITLE,
    (J.NUM_OF_POSITIONS - COUNT(P.JOB_ID))
    FROM DEMOEMPL.JOB J, DEMOEMPL.POSITION P
    WHERE P.FINISH_DATE IS NULL AND P.JOB_ID = J.JOB_ID
    PRESERVE DEMOEMPL.JOB
    GROUP BY J.JOB_ID, J.JOB_TITLE, J.NUM_OF_POSITIONS
    HAVING (J.NUM_OF_POSITIONS - COUNT(P.JOB_ID)) > 0;
```

```
*****
* Create updatable views
*****
CREATE VIEW DEMOEMPL.EMP_HOME_INFO
    AS SELECT EMP_ID, EMP_LNAME, EMP_FNAME, STREET, CITY, STATE,
        ZIP_CODE, PHONE
        FROM DEMOEMPL.EMPLOYEE;

CREATE VIEW DEMOEMPL.EMP_WORK_INFO
    AS SELECT EMP_ID, MANAGER_ID, START_DATE, TERMINATION_DATE
        FROM DEMOEMPL.EMPLOYEE;
```

## B.5 Demo Data

```
*****
INSERT INTO DEMOEMPL.DIVISION
    VALUES ('D02', NULL, 'USED CARS');
INSERT INTO DEMOEMPL.DIVISION
    VALUES ('D04', NULL, 'NEW CARS');
INSERT INTO DEMOEMPL.DIVISION
    VALUES ('D06', NULL, 'SERVICE');
INSERT INTO DEMOEMPL.DIVISION
    VALUES ('D09', NULL, 'CORPORATE');

INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (3510, NULL, 'D02', 'APPRaisal - USED CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (2200, NULL, 'D02', 'SALES - USED CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (1100, NULL, 'D02', 'PURCHASING - USED CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (3520, NULL, 'D04', 'APPRaisal NEW CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (2210, NULL, 'D04', 'SALES - NEW CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (4200, NULL, 'D04', 'LEASING - NEW CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (1110, NULL, 'D04', 'PURCHASING - NEW CARS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (1120, NULL, 'D06', 'PURCHASING - SERVICE');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (4600, NULL, 'D06', 'MAINTENANCE');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (3530, NULL, 'D06', 'APPRaisal - SERVICE');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (5100, NULL, 'D06', 'BILLING');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (6200, NULL, 'D09', 'CORPORATE ADMINISTRATION');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (5200, NULL, 'D09', 'CORPORATE MARKETING');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (5000, NULL, 'D09', 'CORPORATE ACCOUNTING');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (4900, NULL, 'D09', 'MIS');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (6000, NULL, 'D09', 'LEGAL');
INSERT INTO DEMOEMPL.DEPARTMENT
    VALUES (4500, NULL, 'D09', 'HUMAN RESOURCES');
```

```

INSERT INTO DEMOPROJ.PROJECT
    values ('P634', 3411, '2000-02-01', '2000-03-01',
           null, null, 320, null, 'TV ads - WTVK');
INSERT INTO DEMOPROJ.PROJECT
    values ('C200', 3411, '1999-01-15', '2000-04-30', '1999-01-15',
           '2000-04-30', 1776, 2010, 'New brand research');
INSERT INTO DEMOPROJ.PROJECT
    values ('P400', null, '2000-09-01', '2000-12-10',
           null, null, 2960, null, 'Christmas media' );
INSERT INTO DEMOPROJ.PROJECT
    values ('C203', 2894, '1998-02-01', '1998-03-15', '1998-02-10',
           '1998-03-10', 960, 901.50, 'Consumer study' );
INSERT INTO DEMOPROJ.PROJECT
    values ('C240', 4358, '1998-06-01', '1998-07-01', '1998-06-01',
           '1998-08-15', 320, 722.75, 'Service study');
INSERT INTO DEMOPROJ.PROJECT
    values ('D880', 2466, '1999-11-01', '2001-02-01',
           null, null, 960, null, 'Systems analysis' );

INSERT INTO DEMOEMPL.JOB
    values (8001, 'Vice President', 90000, 136000, 'S', 1,
           '1988-01-01',
           'Takes overall responsibility upon president absence',
           'Oversees coordination among divisions and departments');
INSERT INTO DEMOEMPL.JOB
    values (4023, 'Accountant', 44000, 120000, 'S', 1,
           '1985-01-01', 'Responsible for quarterly and final reports',
           'Works with outside consultants on taxes');
INSERT INTO DEMOEMPL.JOB
    values (2051, 'AP Clerk', 8.80, 14.60, 'H', 2,
           '1989-03-01',
           'Responds to incoming invoices by sending out issued checks',
           'Files invoices');
INSERT INTO DEMOEMPL.JOB
    values (2053, 'AR Clerk', 8.80, 14.60, 'H', 3,
           '1989-03-01', 'Sends out customer invoices',
           'Sends out monthly statements and accepts payments');
INSERT INTO DEMOEMPL.JOB
    values (2077, 'Purch Clerk', 17000, 30000, 'S', 3,
           '1989-03-01',
           'Responsible for soliciting quotes from vendors', null);
INSERT INTO DEMOEMPL.JOB
    values (3029, 'Computer Operator', 25000, 44000, 'S', 1,
           '1993-06-01',
           'Responsible for regular operation of computer system',
           'Calls outside maintenance as necessary');

```

```
INSERT INTO DEMOEMPL.JOB
    values (3051, 'Data Entry Clerk', 8.50, 11.45, 'H', 1,
           '1993-06-02', 'Enters A/P and A/R data as necessary',
           null);
INSERT INTO DEMOEMPL.JOB
    values (6011, 'Manager - Acctng', 59400, 121000, 'S', 1,
           '1988-01-01',
           'RESPONSIBILITY FOR ACCOUNTING INCLUDING A/P AND A/R',
           null);
INSERT INTO DEMOEMPL.JOB
    values (4560, 'Mechanic', 11.45, 21.00, 'H', 7,
           '1984-01-01',
           'Works under supervision of senior mechanic to repair cars', null);
INSERT INTO DEMOEMPL.JOB
    values (4666, 'Sr Mechanic', 41000, 91000, 'S', 1,
           '1988-06-01',
           'Oversees maintenance of all cars under warranty or not',
           null);
INSERT INTO DEMOEMPL.JOB
    values (4734, 'Mktng Admin', 25000, 62000, 'S', 2,
           '1994-06-01',
           'Provides marketing plans and ideas for marketing', null);
INSERT INTO DEMOEMPL.JOB
    values (3333, 'Sales Trainee', 21600, 39000, 'S', 4,
           '1994-10-01',
           'Initial sales position for incoming salespeople',
           'Works under supervision of salesperson');
INSERT INTO DEMOEMPL.JOB
    values (5555, 'Salesperson', 30000, 79000, 'S', 9,
           '1984-01-01',
           'Primary responsibility to sell new or used cars', null);
INSERT INTO DEMOEMPL.JOB
    values (6004, 'Manager - HR', 66000, 138000, 'S', 1,
           '1990-06-01',
           'Responsible for hiring, benefits, and education',
           'Also responsible for OSHA compliance');
INSERT INTO DEMOEMPL.JOB
    values (6021, 'Manager - Mktng', 76000, 150000, 'S', 1,
           '1992-01-02',
           'Responsible for all marketing for used and new cars', null);
INSERT INTO DEMOEMPL.JOB
    values (2055, 'PAYROLL CLERK', 17000, 30000, 'S', 1, '1989-03-01',
           'Issue payroll checks to employees and maintains records', null);
INSERT INTO DEMOEMPL.JOB
    values (4025, 'Writer - Mktng', 31000, 50000, 'S', 1,
           '1996-06-01', 'Writes marketing material based on marketingplans',
           null);
INSERT INTO DEMOEMPL.JOB
    values (9001, 'President', 111000, 190000, 'S', 1,
           '1984-01-01', 'Overall responsibility for well-beingof company',
           null);
```

```

INSERT INTO DEMOEMPL.JOB
    values (4123, 'Recruiter', 35000, 56000, 'S', 1,
           '1994-03-01',
           'Posts job openings and submits newspaper ads for openings', null);
INSERT INTO DEMOEMPL.JOB
    values (4130, 'Benefits Analyst', 35000, 56000, 'S', 1,
           '1994-03-01',
           'Maintains benefits information, conforms to govt regulations',
           null);
INSERT INTO DEMOEMPL.JOB
    values (4012, 'Admin Asst', 21000, 44000, 'S', 4,
           '1994-03-01',
           'Assists managers as necessary',
           'Answers phone, files, writes letters, etc.');
INSERT INTO DEMOEMPL.JOB
    VALUES (5111, 'CUST SER REP', 27000, 54000, 'S', 4,
           '1989-06-01',
           'Provides customer support-takes care of complaints',
           'Provides information for customers over the phone');
INSERT INTO DEMOEMPL.JOB
    values (4700, 'Purch Agnt', 33000, 60000, 'S', 5,
           '1993-06-01',
           'Responsible for purchasing decisions for parts and vehicles', null);
INSERT INTO DEMOEMPL.JOB
    values (5890, 'Appraisal Spec', 45000, 70000, 'S', 5,
           '1993-06-01',
           'Responsible for assessing value of vehicles traded in', null);
INSERT INTO DEMOEMPL.JOB
    VALUES (5110, 'CUST SER MGR', 40000, 108000, 'S', 1, '1989-06-01',
           'Responsible for overseeing all customer support', null);

INSERT INTO DEMOPROJ.SKILL
    values (4444, 'Assembly', 'Auto body assembly experience' );
INSERT INTO DEMOPROJ.SKILL
    values (3333, 'Bodywork',
           'Experience in repairing auto bodies' );
INSERT INTO DEMOPROJ.SKILL
    values (3088, 'Brake work', 'Brake diagnosis and repair' );
INSERT INTO DEMOPROJ.SKILL
    values (3065, 'Electronics',
           'Electronic diagnosis and repair' );
INSERT INTO DEMOPROJ.SKILL
    values (1030, 'Acct Mgt',
           'Experience in managing acctng activities' );
INSERT INTO DEMOPROJ.SKILL
    values (5130, 'Basic Math',
           'Knowledge of basic math functions' );
INSERT INTO DEMOPROJ.SKILL
    values (5160, 'Calculus',
           'Knowledge of advanced mathematics' );

```

```
INSERT INTO DEMOPROJ.SKILL
    values (4250, 'Data Entry',
           'Familiarity with computer keyboard' );
INSERT INTO DEMOPROJ.SKILL
    values (4370, 'Filing',
           'Ability to organize correspondence/invoices' );
INSERT INTO DEMOPROJ.SKILL
    values (5200, 'Gen Acctng',
           'Familiarity with basic AR and AP' );
INSERT INTO DEMOPROJ.SKILL
    values (5500, 'Gen Mktng',
           'Knowledge of basic marketing concepts' );
INSERT INTO DEMOPROJ.SKILL
    values (5430, 'Mktng Writing',
           'Background in promotional writing' );
INSERT INTO DEMOPROJ.SKILL
    values (5420, 'Writing', 'General writing skills' );
INSERT INTO DEMOPROJ.SKILL
    values (4490, 'Gen Ledger',
           'Experience with general ledger' );
INSERT INTO DEMOPROJ.SKILL
    values (4430, 'Interviewing',
           'Basic interviewing experience' );
INSERT INTO DEMOPROJ.SKILL
    values (1000, 'Management', 'Experience managing people' );
INSERT INTO DEMOPROJ.SKILL
    values (4420, 'Telephone', 'Basic customer support' );
INSERT INTO DEMOPROJ.SKILL
    values (5180, 'Statistics',
           'Creating & evaluating statistics' );
INSERT INTO DEMOPROJ.SKILL
    values (4410, 'Typing', 'Minimum 60 wpm' );
INSERT INTO DEMOPROJ.SKILL
    values (5309, 'Appraising', 'Used car evaluation' );
INSERT INTO DEMOPROJ.SKILL
    values (6770, 'Purchasing',
           'Basic buying & negotiation procedures' );
INSERT INTO DEMOPROJ.SKILL
    values (7000, 'Sales', 'Background in sales techniques' );
INSERT INTO DEMOPROJ.SKILL
    values (6666, 'Billing', 'Basic billing procedures' );
INSERT INTO DEMOPROJ.SKILL
    values (6650, 'Diesel Engine Repair',
           'Experience in diesel engine repair' );
INSERT INTO DEMOPROJ.SKILL
    values (6670, 'Gas Engine Repair',
           'Experience in gasoline engine repair' );
INSERT INTO DEMOPROJ.SKILL
    values (6470, 'Window Installation',
           'Installation of automotive windows' );
```

```

INSERT INTO DEMOEMPL.EMPLOYEE
    values (1003, null, 'James', 'Baldwin', 6200,
           '21 South St', 'Boston', 'MA', '02010',
           '6173295757', 'A', 076598765, '1984-02-01',
           null, '1951-08-02' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3222, 1003, 'Louise', 'Voltmer', 4500,
           '28 Hayden St', 'Brookline', 'MA', '02066',
           '6176635520', 'A', 090588361, '1993-01-07',
           null, '1968-12-27' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (4321, 1003, 'George', 'Bradley', 6200,
           '344 East Main St', 'Grover', 'MA', '02976',
           '5087463300', 'A', 082999642, '1996-08-04',
           null, '1966-10-31' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (1234, 1003, 'Thomas', 'Mills', 6200,
           '14 Pleasant St', 'Brookline', 'MA', '02066',
           '6176646602', 'A', 055711009, '1985-03-14',
           null, '1969-10-19' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2466, 1003, 'Patricia', 'Bennett', 5000,
           '152B Central St', 'Medford', 'MA', '02432',
           '5089487709', 'A', 098339556, '1991-10-29',
           null, '1963-12-23' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2894, 1003, 'William', 'Griffin', 5200,
           '390 Sherman St', 'Taunton', 'MA', '02678',
           '5088449008', 'A', 077442111, '1992-05-11',
           null, '1966-07-10' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2174, 3222, 'Jonathan', 'Zander', 4500,
           '54 Bradford St', 'Brookline', 'MA', '02066',
           '6176633854', 'A', 032423789, '1997-09-30',
           null, '1969-05-17' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3118, 3222, 'Alan', 'Wooding', 4500,
           '196 School St', 'Canton', 'MA', '02020',
           '5083766984', 'A', 098746783, '1992-11-18',
           null, '1969-05-17' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2461, 1234, 'Alice', 'Anderson', 6200,
           '534 Newton St', 'Medford', 'MA', '02432',
           '5083873664', 'A', 068338909, '1991-09-09',
           null, '1966-07-01' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3841, 2461, 'Michelle', 'Cromwell', 6200,
           '452 Great Rd', 'Boston', 'MA', '02010',
           '6173298763', 'A', 055848876, '1994-10-25',
           null, '1971-05-20' );

```

```

INSERT INTO DEMOEMPL.EMPLOYEE
values (4902, 2461, 'Linda', 'Roy', 6200,
       '29 Westville Ave', 'Wilmington', 'MA', '02476',
       '5088477701', 'A', 098354660, '1995-12-11',
       null, '1972-12-13' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (5103, 2466, 'Adele', 'Ferguson', 5000,
       '12 York Dr', 'Brookline', 'MA', '02066',
       '6176600684', 'A', 095877432, '1999-10-11',
       null, '1977-04-19' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (3449, 2466, 'Cynthia', 'Taylor', 5000,
       '201 Washington St', 'Concord', 'MA', '01342',
       '5082684508', 'A', 088930884, '1993-12-07',
       null, '1968-06-02' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (3411, 2894, 'Catherine', 'Williams', 5200,
       '566 Lincoln St', 'Boston', 'MA', '02010',
       null, 'A', 083356561, '1993-09-30',
       null, '1967-10-28' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (4358, 2894, 'Judith', 'Robinson', 5200,
       '139 White St', 'Wilmington', 'MA', '02476',
       '5087488011', 'A', 075399870, '1996-09-13',
       null, '1964-10-24' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (2781, 4358, 'Joseph', 'Thurston', 5200,
       '4 Birch St', 'Stoneham', 'MA', '02928',
       '6173286008', 'A', 087700466, '1992-04-12',
       null, '1968-11-29' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (2246, 2466, 'Marylou', 'Hamel', 1100,
       '11 Main St', 'Medford', 'MA', '02432',
       '5083457789', 'A', 059975848, '1998-12-07',
       null, '1968-10-24' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (4703, 2246, 'Martin', 'Halloran', 1100,
       '27 Elm St', 'Brookline', 'MA', '02066',
       '6176648290', 'A', 054475888, '1997-03-19',
       null, '1971-12-28' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (5908, 2246, 'Timothy', 'Fordman', 1100,
       '60 Boston Rd', 'Brookline', 'MA', '02066',
       '6176642209', 'A', 033767754, '1998-01-31',
       null, '1973-06-07' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (3082, 2894, 'John', 'Brooks', 3510,
       '129 Bedford St', 'Camden', 'MA', '02113',
       '5089273644', 'A', 098234567, '1992-07-03',
       null, '1970-09-02' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (4773, 3082, 'Janice', 'Dexter', 3510,
       '399 Pine St', 'Medford', 'MA', '02432',
       '5083847566', 'A', 089675632, '1997-06-14',
       null, '1969-11-19' );

```

```

INSERT INTO DEMOEMPL.EMPLOYEE
    values (2180, 2894, 'Joan', 'Albertini', 2200,
            '501 Piper Rd', 'Medford', 'MA', '02432',
            '5083145366', 'A', 066783225, '1989-10-27',
            null, '1964-03-26' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (4660, 2180, 'Bruce', 'MacGregor', 2200,
            '254 Waterside Rd', 'Camden', 'MA', '02113',
            '5092344620', 'A', 098363389, '1997-01-20',
            null, '1965-10-28' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3767, 2180, 'Frank', 'Lowe', 2200,
            '25 Rutland St', 'Natick', 'MA', '02364',
            '5082844094', 'A', 066985009, '1994-08-31',
            null, '1964-12-08' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2448, 2180, 'David', 'Lynn', 2200,
            '93 Hubbard St', 'Natick', 'MA', '02364',
            '5082844736', 'A', 028448958, '1991-09-01',
            null, '1961-03-02' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3704, 2448, 'Richard', 'Moore', 2200,
            '130 Swanson St', 'Dedham', 'MA', '02026',
            '6177739440', 'A', 095435467, '1994-04-10',
            null, '1961-11-23' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (1765, 2466, 'David', 'Alexander', 1110,
            '18 Cross St', 'Grover', 'MA', '02976',
            '5087394772', 'A', 048903743, '1985-10-23',
            null, '1955-11-13' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2106, 1765, 'Susan', 'Widman', 1110,
            '43 Oak St', 'Medford', 'MA', '02432',
            '5083346364', 'A', 109857893, '1989-05-01',
            null, '1971-05-11' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3769, 2894, 'Julie', 'Donelson', 3520,
            '14 Atwood Rd', 'Grover', 'MA', '02976',
            '5084850432', 'A', 067783532, '1994-08-31',
            null, '1967-08-15' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2010, 2894, 'Cora', 'Parker', 2210,
            '2 Spring St', 'Boston', 'MA', '02010',
            null, 'A', 086574983, '1988-03-18',
            null, '1962-05-25' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (4001, 2010, 'Jason', 'Thompson', 2210,
            '3 Flintlock St', 'Natick', 'MA', '02364',
            '5082649956', 'A', 054578957, '1995-12-11',
            null, '1964-08-15' );

```

```
INSERT INTO DEMOEMPL.EMPLOYEE
values (4908, 2010, 'Robert', 'Clark', 2210,
       '54 Tenny St', 'Brookline', 'MA', '02066',
       null, 'A', 198546272, '1996-01-23',
       null, '1959-11-01');
INSERT INTO DEMOEMPL.EMPLOYEE
values (4962, 2010, 'Peter', 'White', 2210,
       '1440 Mass Ave', 'Boston', 'MA', '02010',
       '6177732280', 'A', 123395857, '1997-10-04',
       null, '1959-07-01');
INSERT INTO DEMOEMPL.EMPLOYEE
values (3764, 2010, 'Deborah', 'Park', 2210,
       '379 Center St', 'Brookline', 'MA', '02066',
       '6179458377', 'A', 034222564, '1994-08-25',
       null, '1960-03-08');
INSERT INTO DEMOEMPL.EMPLOYEE
values (5090, 2010, 'Stephen', 'Wills', 2210,
       '34 Avon Dr', 'Canton', 'MA', '02020',
       '5083389935', 'A', 012434452, '1998-07-12',
       null, '1972-04-25');
INSERT INTO DEMOEMPL.EMPLOYEE
values (3991, 2010, 'Fred', 'Wilkins', 2210,
       '344 Stevens St', 'Taunton', 'MA', '02678',
       '5081840883', 'A', 026475929, '1994-11-12',
       null, '1963-03-29');
INSERT INTO DEMOEMPL.EMPLOYEE
values (4027, 3991, 'Cecile', 'Courtney', 2210,
       '99 West Main St', 'Natick', 'MA', '02364',
       '5089445386', 'A', 012209982, '1996-04-01',
       null, '1967-07-07');
INSERT INTO DEMOEMPL.EMPLOYEE
values (3778, 2466, 'Jane', 'Ferndale', 5100,
       '15 Dawson St', 'Medford', 'MA', '02432',
       '6173450099', 'A', 10477822, '1994-09-07',
       null, '1962-11-30');
INSERT INTO DEMOEMPL.EMPLOYEE
values (2598, 3778, 'Mary', 'Jacobs', 5100,
       '24A Main St', 'Camden', 'MA', '02113',
       null, 'A', 339000022, '1992-01-03',
       null, '1974-05-02');
INSERT INTO DEMOEMPL.EMPLOYEE
values (2004, 2466, 'Eleanor', 'Johnson', 1120,
       '225 Fisk St', 'Medford', 'MA', '02432',
       '5089253998', 'A', 01010885, '1988-02-28',
       null, '1952-12-23');
INSERT INTO DEMOEMPL.EMPLOYEE
values (3294, 2004, 'Carolyn', 'Johnson', 1120,
       '79 High St', 'Brookline', 'MA', '02066',
       '6175567551', 'A', 038800922, '1993-02-19',
       null, '1967-10-05');
```

```

INSERT INTO DEMOEMPL.EMPLOYEE
    values (3338, 2004, 'Mark', 'White', 1120,
           '560 Camden St', 'Canton', 'MA', '02020',
           '6179238844', 'A', 055002432, '1993-07-02',
           null, '1964-08-15' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2209, 2894, 'Michael', 'Smith', 3530,
           '201 Summer St', 'Brookline', 'MA', '02066',
           '6175563331', 'A', 093666540, '1990-06-17',
           null, '1959-12-13' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3341, 2209, 'Carl', 'Smith', 3530,
           '18 South St', 'Newton', 'MA', '02576',
           '6179658099', 'A', 033970385, '1993-07-02',
           null, '1962-02-03' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2096, 4321, 'Thomas', 'Carlson', 4600,
           '23 Hemmingway Ln', 'Brookline', 'MA', '02066',
           '6175553643', 'A', 041783445, '1989-01-26',
           null, '1964-04-14' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2437, 2096, 'Henry', 'Thompson', 4600,
           '1467 West Ave', 'Boston', 'MA', '02030',
           '6179264105', 'S', 44622905, '1991-08-06',
           null, '1966-10-12' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3433, 2096, 'Herbert', 'Crane', 4600,
           '20 W Bloomfield Ave', 'Newton', 'MA', '02456',
           '6178653440', 'A', 209338445, '1993-11-01',
           null, '1958-05-30' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (1034, 2096, 'James', 'Gallway', 4600,
           '12 East Speen St', 'Stoneham', 'MA', '02928',
           '6172251178', 'A', 067775312, '1984-02-01',
           null, '1951-11-23' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2424, 1034, 'Ronald', 'Wilder', 4600,
           '30 Heron Ave', 'Natick', 'MA', '02178',
           '5083347700', 'A', 056668338, '1991-07-24',
           null, '1948-09-09' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (4456, 1034, 'Thomas', 'Thompson', 4600,
           '32 South Broadway', 'Newton', 'MA', '02576',
           '6179660089', 'A', 077492347, '1997-01-04',
           null, '1978-09-13' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (3288, 1034, 'Ralph', 'Sampson', 4600,
           '99 Vale Ave', 'Newton', 'MA', '02576',
           '6179654443', 'A', 077447333, '1993-01-29',
           null, '1962-09-30' );
INSERT INTO DEMOEMPL.EMPLOYEE
    values (2299, null, 'Samuel', 'Spade', 4600,
           '47 London St', 'Canton', 'MA', '02020',
           null, 'L', 033892200, '1991-02-04',
           null, '1958-01-09' );

```

```
INSERT INTO DEMOEMPL.EMPLOYEE
values (3199, null, 'Martin', 'Loren', 4600,
       '401 Cross St', 'Grover', 'MA', '02976',
       null, 'L', 098884332, '1992-12-05',
       null, '1962-10-19' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (2145, null, 'Martin', 'Catlin', 5200,
       '44 Smithville Hts', 'Wilmington', 'MA', '02476',
       '5087486625', 'L', 044895224, '1989-09-24',
       null, '1954-03-02' );
INSERT INTO DEMOEMPL.EMPLOYEE
values (2898, null, 'Mary', 'Umidy', 1120,
       '895A Braintree Circle', 'Medford', 'MA', '02432',
       '6173458860', 'S', 056906868, '1992-05-11',
       null, '1962-05-11' );
INSERT INTO DEMOEMPL.POSITION
values (4773, 5890, '1997-06-14', null, null, 45240.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (1234, 8001, '1985-03-14', null, null, 117832.68,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (3082, 5890, '1992-07-03', null, null, 68016.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (2180, 5555, '1990-04-18', null, null, 76961.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (4660, 5555, '1997-03-31', null, null, 36400.00,
       .25, .157, null );
INSERT INTO DEMOEMPL.POSITION
values (3767, 5555, '1995-01-11', null, null, 50440.50,
       .23, .125, null );
INSERT INTO DEMOEMPL.POSITION
values (2448, 5555, '1991-09-01', null, null, 70720.00,
       .255, .157, null );
INSERT INTO DEMOEMPL.POSITION
values (3704, 3333, '1994-04-10', null, null, 22880.00,
       null, .105, null );
INSERT INTO DEMOEMPL.POSITION
values (4703, 2077, '1997-03-19', null, null, 24857.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (2246, 4700, '1993-09-28', null, null, 59488.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (5008, 4700, '1998-01-31', null, null, 47944.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (3769, 5890, '1994-08-31', null, null, 41600.00,
       null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (4001, 5555, '1995-12-11', null, null, 36921.00,
       .23, .125, null );
```

```
INSERT INTO DEMOEMPL.POSITION
    values (4008, 3333, '1996-01-23', null, null, 24441.00,
            null, .99, null );
INSERT INTO DEMOEMPL.POSITION
    values (4962, 3333, '1997-10-04', null, null, 30680.00,
            null, .125, null );
INSERT INTO DEMOEMPL.POSITION
    values (2010, 5555, '1988-03-18', null, null, 76440.00,
            .275, .180, null );
INSERT INTO DEMOEMPL.POSITION
    values (3764, 5555, '1995-10-02', null, null, 54184.00,
            .26, .170, null );
INSERT INTO DEMOEMPL.POSITION
    values (5090, 5555, '1998-07-12', null, null, 48568.48,
            .205, .135, null );
INSERT INTO DEMOEMPL.POSITION
    values (4027, 3333, '1996-04-01', null, null, 28081.40,
            null, .120, null );
INSERT INTO DEMOEMPL.POSITION
    values (3991, 5555, '1995-06-06', null, null, 42016.00,
            .235, .125, null );
INSERT INTO DEMOEMPL.POSITION
    values (1765, 4700, '1992-06-10', null, null, 47009.34,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2106, 2077, '1989-05-01', null, null, 23920.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2096, 4666, '1994-10-10', null, null, 85280.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2437, 4560, '1991-08-06', null, 14.55, null,
            null, null, 21.83 );
INSERT INTO DEMOEMPL.POSITION
    values (2598, 2053, '1992-01-03', null, 10.50, null,
            null, null, 15.00 );
INSERT INTO DEMOEMPL.POSITION
    values (3433, 4560, '1993-11-01', null, 19.15, null,
            null, null, 28.00 );
INSERT INTO DEMOEMPL.POSITION
    values (3778, 2053, '1994-09-07', null, 9.98, null,
            null, null, 14.00 );
INSERT INTO DEMOEMPL.POSITION
    values (1034, 4560, '1984-02-01', null, 20.93, null,
            null, null, 29.50 );
INSERT INTO DEMOEMPL.POSITION
    values (2424, 4560, '1991-07-24', null, 13.60, null,
            null, null, 19.40 );
INSERT INTO DEMOEMPL.POSITION
    values (2004, 4700, '1993-11-19', null, null, 59280.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (4456, 4560, '1997-01-04', null, 14.58, null,
            null, null, 19.87 );
```

```
INSERT INTO DEMOEMPL.POSITION
    values (3288, 4560, '1993-01-29', null, 16.40, null,
           null, null, 23.60 );
INSERT INTO DEMOEMPL.POSITION
    values (3341, 5890, '1993-07-02', null, null, 48465.80,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2209, 5890, '1990-06-17', null, null, 66144.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3294, 4700, '1993-02-19', null, null, 53665.56,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3338, 2077, '1993-07-02', null, null, 22048.84,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2174, 4123, '1989-09-30', null, null, 49921.76,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3118, 4130, '1992-11-18', null, null, 45241.94,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3222, 6004, '1993-01-07', null, null, 110448.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (4321, 5110, '1996-08-04', null, null, 56977.80,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2461, 4012, '1991-09-09', null, null, 43784.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3841, 4012, '1994-10-25', null, null, 33800.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (4002, 4012, '1995-12-11', null, null, 28601.80,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (1003, 9001, '1984-02-01', null, null, 146432.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (5103, 2051, '1999-10-11', null, 7.13, null,
           null, null, 11.70 );
INSERT INTO DEMOEMPL.POSITION
    values (2466, 6011, '1991-10-29', null, null, 94953.52,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3449, 4023, '1993-12-07', null, null, 74776.00,
           null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2781, 4025, '1992-04-12', null, null, 43888.00,
           null, null, null );
```

```

INSERT INTO DEMOEMPL.POSITION
    values (2894, 6021, '1992-05-11', null, null, 111593.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3411, 4734, '1995-04-02', null, null, 53665.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (4358, 4734, '1996-09-13', null, null, 57824.50,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3764, 3333, '1994-08-25', '1995-10-01', NULL, 28912.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (3991, 3333, '1994-11-12', '1995-06-05', null, 27976.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2246, 2077, '1990-12-07', '1993-09-27', null, 29536.00,
            null, null, null );
INSERT INTO DEMOEMPL.POSITION
    values (2096, 4560, '1989-01-26', '1994-10-09', 17.90, null,
            null, null, 28.85 );
INSERT INTO DEMOEMPL.POSITION
    values (3767, 3333, '1994-08-31', '1995-01-10', null, 2200.00,
            null, .105, null);
INSERT INTO DEMOEMPL.POSITION
    values (2180, 3333, '1997-10-27', '1990-04-17', null, 19000.10,
            null, .09, null);
INSERT INTO DEMOEMPL.POSITION
    values (4660, 3333, '1997-01-20', '1997-03-30', null, 24000.00,
            null, .11, null);
INSERT INTO DEMOEMPL.POSITION
    values (1765, 2077, '1985-10-23', '1992-06-10', null, 18001.00,
            null, null, null);
INSERT INTO DEMOEMPL.POSITION
    values (2004, 2053, '1988-02-28', '1993-11-18', 9.50, null,
            null, null, 13.50 );
INSERT INTO DEMOEMPL.POSITION
    values (3411, 4012, '1993-09-30', '1995-04-01', null, 44001.40,
            null, null, null);
INSERT INTO DEMOPROJ.EXPERTISE
    values (4773, 5309, '02', '1995-10-14' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (1234, 1000, '04', '1988-06-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3082, 5309, '04', '1994-06-03' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2180, 7000, '04', '1993-01-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4660, 7000, '03', '1995-10-09' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3767, 7000, '04', '1994-09-20' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2448, 7000, '03', '1991-06-10' );

```

```
INSERT INTO DEMOPROJ.EXPERTISE
    values (3704, 7000, '01', '1993-08-21' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4703, 4250, '03', '1996-11-20' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2246, 1000, '03', '1993-10-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2246, 6670, '04', '1990-03-29' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (5008, 6770, '04', '1998-01-31' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4703, 5130, '03', '1998-03-30' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3769, 5309, '04', '1992-10-04' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4001, 7000, '03', '1994-12-11' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4008, 4420, '01', '1994-12-14' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4962, 5130, '02', '1992-11-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2010, 7000, '03', '1988-02-18' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3764, 7000, '03', '1992-01-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (5090, 7000, '03', '1997-02-12' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4027, 7000, '01', '1995-03-19' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3991, 7000, '03', '1995-01-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (1765, 6770, '04', '1985-10-23' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2106, 6770, '03', '1991-10-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2096, 3333, '02', '1995-03-03' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2096, 3065, '03', '1998-04-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2437, 3333, '04', '1995-03-15' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2437, 4444, '04', '1997-05-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2598, 6666, '03', '1997-07-25' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3433, 6650, '02', '1991-10-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3778, 5200, '03', '1998-01-21' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3778, 6666, '04', '1998-05-15' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (1034, 6470, '02', '1984-02-21' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2424, 6470, '03', '1989-04-18' );
```

```
INSERT INTO DEMOPROJ.EXPERTISE
    values (2004, 6770, '04', '1988-02-28' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4456, 6670, '01', '1993-06-02' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4456, 3065, '02', '1993-09-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3288, 6650, '02', '1993-06-12' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3288, 6670, '01', '1994-12-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3288, 3333, '04', '1993-12-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3341, 5309, '03', '1993-10-02' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2209, 5309, '04', '1992-08-12' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3294, 6770, '01', '1989-09-21' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3338, 6770, '03', '1994-12-11' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2174, 4430, '04', '1995-03-30' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3118, 5180, '03', '1995-07-23' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3222, 1000, '04', '1995-10-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3222, 4430, '04', '1996-12-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4321, 4430, '04', '1997-03-24' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4321, 1000, '03', '1998-06-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2461, 4370, '04', '1994-03-12' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2461, 4250, '04', '1997-03-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2461, 5180, '03', '1997-06-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3841, 4370, '03', '1995-10-10' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3841, 4410, '02', '1996-06-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4002, 4370, '03', '1996-02-15' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4002, 4410, '04', '1999-01-15' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (1003, 1000, '04', '1984-02-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (5103, 5200, '04', '1997-10-11' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2466, 1030, '04', '1991-10-29' );
```

```

INSERT INTO DEMOPROJ.EXPERTISE
    values (2466, 5200, '04', '1999-06-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2466, 4490, '03', '1999-12-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3449, 5200, '03', '1993-09-29' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2781, 5430, '01', '1995-09-27' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2781, 5420, '02', '1996-12-01' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2894, 1000, '04', '1995-11-12' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (2894, 5500, '04', '1996-12-15' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (3411, 5500, '04', '1997-01-30' );
INSERT INTO DEMOPROJ.EXPERTISE
    values (4358, 5500, '03', '1996-12-30' );
INSERT INTO DEMOPROJ.CONULTANT
    values (9443, 'Diane', 'Jones', 2466, 5200, 'D880',
           '183 Hawthorne Ln', 'Medford', 'MA', '02432',
           '5084475583', '1957-01-23', '1999-08-08', 089393334,
           50.00 );
INSERT INTO DEMOPROJ.CONULTANT
    values (9439, 'Charles', 'Miller', 2466, 4900, 'D880',
           '85 St. James St', 'Brookline', 'MA', '02066',
           '6174800873', '1963-09-12', '1999-02-18', 085763854,
           47.00 );
INSERT INTO DEMOPROJ.CONULTANT
    values (9388, 'Linda', 'Candido', 2466, 5200, 'D880',
           '54 Church St', 'Newton', 'MA', '02456',
           '6179943082', '1959-08-30', '1997-12-21', 033006132,
           76.00 );
INSERT INTO DEMOPROJ.CONULTANT
    values (9000, 'James', 'Legato', 1003, 6000, null,
           '85 North Rd', 'Newton', 'MA', '02456',
           '6179964874', '1970-05-20', '1994-03-20', 095578460,
           148.00 );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 4773, 68, 68, 8.00, 5.00, 0 ,0
           , '2000-10-15', .05, null,
           NULL, NULL, 900.00, 0 ,0,
           'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 3082, 68, 52, 8, 8, 0 ,0
           , '2000-10-20', .055, null,
           '401K', .08, 1400.00, 0 ,0,
           'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2180, 92.50, 0, 8.00, 4.00, 0 ,0
           , '2000-10-30', .06, null,
           'STOCK', .05, 2100.00, 16, 0 ,
           'COLL', null, null);

```

```

INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 4660, 68, 56, 8.00, 0, .07,
            3095, '2000-01-13', .06, null,
            '401K', .05, 850.68,0,0,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3767, 68, 68, 8.00, 0, .07,
            2250, '2000-09-22', .045, null,
            '401K', .05, 1350.50, 16, 16,
            'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 2448, 68, 20.50, 8.00, 3, .075,
            6600, '2000-07-13', .05, null,
            'BONDS', .08, 2100.00, 0,0,
            'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3704, 68, 48, 8.00, 8.00, .05,
            3470, '2000-04-30', .045, null,
            'BONDS', .04, 1800.00, 8, 8,
            'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 4703, 46.75, 16, 8.00, 14.5, .05,
            3010, '2000-03-10', .08, null,
            NULL, NULL, 1107.50,0,0,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 2246, 92.50, 72, 8.00, 5, .05,
            4500, '2000-12-15', .08, '1993-09-27',
            null, null, 2300.00, 24.5, 16.00,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 5008, 46.5, 40, 8.00, 0, .10,
            2000, '2000-01-29', .06, null,
            '401K', .05, 307.50,0,0,
            'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3769, 68, 0, 8.00, 6.00, .10,
            6600, '2000-10-01', .04, null,
            '401K', .03, 1356.70,0,0,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 4001, 68, 40, 8.00, 2.5, 0,0
            , '2000-12-20', .04, null,
            NULL, NULL, 1756.50,0,0,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 4008, 68, 0, 8.00, 3.5,0,0
            , '2000-01-14', .05, null,
            '401K', .05, 1750.00,0,0,
            'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 4962, 68, 16, 8.00, 7.5, 0,0
            , '2000-10-04', .06, null,
            '401K', .06, 1307.80, 8.5, 8.5,

```

```
'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2010, 92.75, 16.00, 8.00, 2.5,0,0
        , '2000-03-18', .05, null,
        'STOCK', .05, 2450.50, 0,0,
        'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3764, 68, 80, 8.00, 5.00, .08,
        3060, '2000-06-11', .065, '1991-05-10',
        'STOCK', .06, 1406.90, 32.5, 16.0,
        'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 5090, 46, 0, 8.00, 0,0,0
        , '2000-07-14', .04, null,
        NULL, NULL, 0,0,0,
        'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 4027, 68, 40, 8.00, 4.00, .08,
        3000, '2000-07-19', .035, null,
        '401K', .04, 1750.00,0,0,
        'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3991, 68, 68, 8.00, 3.00, .08,
        4500, '2000-11-12', .055, '1995-06-05',
        '401K', .06, 1354.60, 8.0, 0,
        'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 1765, 92.5, 32, 8.00, 0, .10,
        7600, '2000-10-23', .07, null,
        '401K', .08, 2500.00, 32, 0,
        'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 2106, 92.5, 32, 8.00, 1.00, .08,
        5500, '2000-04-16', .06, '1999-08-17',
        'BONDS', .04, 2100.00, 0,0,
        'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2096, 92.5, 80, 8.00, 5.00, .05,
        5300, '2000-02-28', .055,
        '1998-10-09','STOCK', .05, 2300.00, 0,0,
        'HSDIP', NULL, NULL);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2437, 68, 0, 8.00, 4.5, 0,0
        , '2000-08-16', .04, null,
        NULL, NULL, 2100.00, 0,0,
        'GED', 'MC655-6901', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000,2598, 60, 8, 20.00, 8.5, 0 ,0
        , '2000-01-26', .035, null,
        NULL, NULL, 2300.00, 0,0,
        'HSDIP', 'HP302-7409', 50.50);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 3433, 68, 40, 8.00, 4.00,0,0
      , '2000-10-23', .05, null,
      NULL, NULL, 1456.70,0,0,
      'JRCOLL', 'MC655-7487', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 3778, 68, 40, 8.00, 4,0,0
      , '2000-09-24', .06, null,
      NULL, NULL, 1350.50,0,0,
      'HSDIP', 'HP302-7487', 50.50);
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 1034, 92.5, 72, 8.00, 2.5, .10,
      5540, '2000-01-24', .05, null,
      'BONDS', .06, 2900.00, 0,0,
      'HSDIP', 'MC655-4490', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 2424, 92.5, 48, 8.00, 3.5, .05,
      2460, '2000-07-19', .04, null,
      NULL, NULL, 2100.00, 0,0,
      'HSDIP', 'MC655-5571', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 2004, 92.5, 40, 8.00, 0, .05,
      2300, '2000-02-28', .03, null,
      '401K', .04, 2450.50,0,0,
      'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 4456, 68, 40, 8.00, 7.00,0,0
      , '2000-01-05', .03, null,
      NULL, NULL, 906.50,0,0,
      'HSDIP', 'MC655-6680', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 3288, 68, 56, 8.00, 2.00,0,0
      , '2000-01-05', .04, null,
      NULL, NULL, 1500.00, 0,0,
      'HSDIP', 'MC655-4402', 90.55);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 3341, 68, 32.5, 8.00, 3.00,0,0
      , '2000-10-05', .045, null,
      '401K', .07, 1500.00, 0,0,
      'COLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 2209, 92.50, 32, 8.00, 5.5,0,0
      , '2000-06-14', .06, null,
      '401K', .06, 2300.00, 16.00, 16.00,
      'COLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
VALUES (2000, 3294, 68, 16, 8.00, 3.00,0,0
      , '2000-02-28', .055, null,
      '401K', .03, 1500.00, 0,0,
      'COLL', null, null );
```

```
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 3338, 68, 0, 8.00, 1.5,0,0
        , '2000-07-02', .05, null,
        NULL, NULL, 1450.50,0,0,
        'HSDIP', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2174, 92, 48, 8.00, 9.00,0,0
        , '2000-09-27', .06, null,
        '401K', .04, 2100.00, 0,0,
        'JRCOLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3118, 68, 8, 8.00, 7.00, .05,
        2010, '2000-11-24', .045, null,
        'BONDS', .08, 1500.00, 8.5, 8.00,
        'COLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3222, 68, 0, 8.00, 2.5, .05,
        2240, '2000-01-02', .07, '1999-06-08',
        '401K', .09, 1350.50, 32, 8,
        'MAS', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 4321, 68, 48, 8.00, 3.00, .05,
        1991, '2000-08-02', .05, null,
        NULL, NULL, 1200.00, 0,0,
        'JRCOLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2461, 68, 40, 8.00, 1.5,0,0
        , '2000-09-13', .04, null,
        NULL, NULL, 2100.00,0 ,0,
        'HSDIP', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 3841, 68, 0, 8.00, 2.00,0,0
        , '2000-10-10', .06, null,
        NULL, NULL, 1300.00, 0,0,
        'JRCOLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 4002, 68, 40, 8.00, 4.5,0,0
        , '2000-12-15', .045, null,
        NULL, NULL, 1750.50,0,0,
        'HSDIP', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 1003, 92, 0, 8.00, 0, .10,
        12340, null, .05, null,
        '401K', .10, NULL,0,0,
        'MAS', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 5103, 46, 0, 8, 0, .05,
        530, '2000-10-11', .05, null,
        NULL, NULL, NULL,0,0,
        'HSDIP', 'HP302-8403', 50.50);
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 2466, 92.5, 40, 8.00, 3.5, .05,
        3400, '2000-10-30', .055, null,
        '401K', .05, 2100.00, 16, 16,
        'COLL', null, null );
```

```

INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 3449, 68, 56, 8.00, 10.5, .07,
            3700, '2000-12-02', .045, null,
            '401K', .03, 1453.70,0,0,
            'COLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2781, 68, 60, 8.00, 7.00,0,0
            , '2000-04-25', .05, null,
            '401K', .03, 2105.90,0,0,
            'COLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 2894, 68, 0, 8.00, 2.5,0,0
            , '2000-05-04', .055, null,
            'STOCK', .08, 2155.30, 16.5, 8,
            'MAS', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (2000, 3411, 68, 68, 8, 8,0,0
            , '2000-09-30', .05, NULL,
            '401K', .03, 1400.00, 0,0,
            'JRCOLL', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    values (2000, 4358, 68, 0, 8.00, 6.5, .07,
            1430, '2000-09-27', .055, null,
            NULL, NULL, 950.50,0,0,
            'HSDIP', null, null );
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 4773, 80, 80, 15, 1, 0 ,0, '1999-07-02',
            .04, NULL, NULL, NULL, 600.00, 0,0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 4773, 80, 48, 10, 10, 0 ,0, '1998-07-05',
            .03, NULL, NULL, NULL, 500.00, 0,0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 4773, 24, 24, 4.5, 0, 0 , 0 , NULL, NULL,
            NULL, NULL, NULL,0,0, 'COLL', NULL,
            null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 3082, 120, 120, 15, 8, 0 ,0, '1999-10-12',
            .05, NULL, NULL, NULL, 1100.00, 0,0, 'JRCOLL', NULL,
            null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3082, 120, 120, 15, 4.5, 0 ,0,
            '1998-01-09',
            .05, NULL, NULL, NULL, 1000.00,0 ,0, 'JRCOLL', NULL,
            null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3082, 120, 120, 15, 2, 0 , 0 , '1997-10-01',
            .05, NULL, NULL, NULL, 1000.00, 0,0, 'JRCOLL',
            null, null);

```

```

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2180, 160, 160, 15, 6, 0 ,0, '1999-10-17',
            .05, NULL, 'STOCK', .05, 2000.00, 0,0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2180, 120, 120, 15, 2.5, 0 ,0, '1998-10-25',
            .055, NULL, 'STOCK', .05, 1900.00, 0,0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2180, 120, 120, 15, 7, 0 ,0, '1997-10-02',
            .05, NULL, 'STOCK', .05, 2000.00, 0,0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 4660, 80, 80, 15, 10, .05, 2060, '1999-01-15',
            .055, NULL, '401K', .05, 750.60, 0 ,0 , 'HSDIP',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 4660, 80, 80, 10, 5, 0 ,0, '1998-01-30',
            .04, NULL, '401K', .05, 500.00, 0 ,0 , 'HSDIP', NULL,
            null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 4660, 48, 48, 8, 2.5, 0 ,0 , NULL, NULL,
            NULL, '401K', .04, 400.00, 0,0, 'HSDIP',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 3767, 120, 120, 15, 0, .07, 2400, '1999-08-17',
            .05, NULL, '401K', .05, 1000.00, 0,0, 'JRCOLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 3767, 120, 120, 15, 0, .07, 2200, '1998-08-10',
            .05, NULL, '401K', .05, 1000.00, 0,0,
            'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1997, 3767, 120, 120, 15, 0, .07, 2000, '1997-08-01',
            .05, NULL, '401K', .05, 1350.00, 0,0, 'JRCOLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2448, 120, 120, 15, 8,0 ,0, '1999-09-18',
            .04, NULL, 'BONDS', .08, 1700.00, 0 ,0 , 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2448, 120, 120, 15, 15, 0,0, '1998-09-15',
            .035, NULL, 'BONDS', .08, 1500.00, 0,0 , 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2448, 120, 120, 15, 5, 0,0, '1997-08-30',
            .03, NULL, 'BONDS', .08, 1500.00, 0 ,0 , 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 3704, 120, 120, 15, 15, .04, 2800, '1999-04-24',
            .045, NULL, 'BONDS', .04, 1700.00, 0,0, 'JRCOLL',
            null, null);

```

```

INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 3704, 120, 120, 15, 15, .03, 2200, '1998-04-30',
        .04, NULL, 'BONDS', .04, 1500.00, 0,0, 'JRCOLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3704, 120, 120, 15, 15, 0,0, '1997-04-20',
        .035, null, null, null, 1300.00, 12, 12, 'JRCOLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 4703, 80, 80, 15, 1, .04, 2300, '1999-03-10',
        .065, NULL, NULL, NULL, 950.00,0,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 4703, 80, 80, 10, 2.5, .04, 2010, '1998-03-30',
        .05, NULL, NULL, NULL, 800.00,0 ,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 4703, 36, 36, 6, 0, 0 ,0 , NULL, NULL,
        NULL, NULL, NULL,0,0, 'HSDIP', NULL,
        null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 2246, 160, 160, 15, 3, .04, 3500, '1999-12-06',
        .07, '1993-09-27', NULL, NULL, 2100.00,0 ,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2246, 120, 120, 15, 3,0 ,0, '1998-12-01',
        .065, '1993-09-27', NULL, NULL, 1700.00, 0,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2246, 120, 120, 15, 5,0 ,0, '1997-12-20',
        .06, '1993-09-27', NULL, NULL, 1600.00, 0,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 5008, 80, 80, 10, 6, .10, 1700, '1999-02-07',
        .04, NULL, NULL, NULL, 200.00, 0,0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 5008, 48, 48, 8, 7, .10, 1500, null, null,
        NULL, '401K', .05, NULL,0,0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 3769, 120, 120, 15, 14,0 ,0, '1999-09-17',
        .04, NULL, '401K', .03, 1200.00,0 ,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3769, 120, 120, 15, 8.5,0 ,0, '1998-09-01',
        .04, NULL, '401K', .04, 1100.00,0 ,0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3769, 120, 120, 15, 3, 0,0, '1997-09-06',
        .04, NULL, '401K', .04, 1000.00,0 ,0, 'HSDIP',
        null, null);

```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4001, 120, 120, 15, 3, 0 ,0, '1999-12-01',
       .045, NULL, NULL, NULL, 1500.00,0 ,0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4001, 80, 80, 15, 8, 0 ,0, '1998-12-18',
       .04, NULL, NULL, NULL, 1200.00,0 ,0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4001, 80, 80, 15, 3, 0,0, '1997-12-10',
       .04, NULL, NULL, NULL, 1000.00, 0,0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4008, 120, 120, 15, 2, 0,0, '1999-01-15',
       .04, NULL, '401K', .05, 1500.00,0 ,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4008, 80, 80, 15, 1, 0,0, '1998-01-31',
       .035, NULL, '401K', .05, 1350.00,0,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4008, 80, 72, 15, 0, 0,0, '1997-01-30',
       .035, NULL, NULL, NULL, 1100.00,0 ,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4962, 80, 80, 15, 4.5,0 ,0, '1999-10-10',
       .06, NULL, '401K', .05, 1150.50,0,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4962, 80, 80, 10, 1,0 ,0, '1998-10-16',
       .05, null, '401K', .05, 1000.00, 2, 2, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4962, 12, 0, 2, 0, .05, 3000, null, null,
       NULL, NULL, NULL, NULL,0,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2010, 160, 160, 15, 4,0 ,0, '1999-03-01',
       .055, NULL, 'STOCK', .05, 2100.00,0 ,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2010, 160, 152.5, 15, 3, 0,0, '1998-03-30',
       .05, NULL, 'STOCK', .05, 2000.00,0 ,0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2010, 160, 160, 15, 3,0 ,0, '1997-03-10',
       .05, null, 'BONDS', .05, 1600.00, 2, 2, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3764, 120, 120, 15, 2, 0,0, '1999-08-01',
       .055, '1991-05-10', 'STOCK', .06, 1500.00,0 ,0,
       'COLL', null, null);
```

```

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3764, 120, 120, 15, 3, 0, 0, '1998-08-30',
            .05, '1991-05-10', 'STOCK', .05, 1200.00, 14, 14,
            'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3764, 120, 120, 15, 5, 0, 0, '1997-08-17',
            .045, '1991-05-10', 'STOCK', .05, 1000.00, 0, 0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 5090, 80, 80, 15, 2, 0, 0, '1999-07-30',
            .035, null, null, null, 800.00, 0, 0, 'JRCOLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 5090, 24, 24, 4, 2, 0, 0, null, null,
            null, null, null, 0, 0, 'JRCOLL',
            null, null);

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 4027, 120, 120, 15, 8, 0, 0, '1999-03-15',
            .03, null, null, null, 1500.00, 16, 16, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 4027, 120, 120, 15, 0, 0, 0, '1998-04-30',
            .03, null, null, null, 1200.00, 0, 0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 4027, 80, 80, 10, 2.5, 0, 0, '1997-04-01',
            .03, null, null, null, 1000.00, 0, 0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 3991, 120, 120, 15, 8, .08, 4000, '1999-12-04',
            .05, '1995-06-05', '401K', .05, 1300.00, 0, 0, 'COLL',
            null, null);

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3991, 120, 116, 15, 2, 0, 0, '1998-11-28',
            .045, '1995-06-05', '401K', .05, 1100.00, 8, 8, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3991, 120, 120, 15, 8, 0, 0, '1997-11-30',
            .045, '1995-06-05', null, null, 1000.00, 0, 0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 1765, 160, 160, 15, 0, .10, 7000, '1999-11-15',
            .07, null, '401K', .08, 2500.50, 36, 0, 'COLL',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 1765, 160, 160, 15, 0, .10, 6500, '1998-11-01',
            .07, null, '401K', .08, 2500.00, 88, 0, 'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1997, 1765, 160, 160, 15, 0, .10, 6000, '1997-10-30',
            .065, null, '401K', .07, 2400.00, 72, 0, 'COLL',
            null, null);

```

```

INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 2106, 160, 160, 15, 9.5, .07, 4500, '1999-05-01',
            .055, '1999-08-17', 'BONDS', .04, 1800.00, 0 , 0 ,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2106, 160, 160, 15, 3,0 ,0, '1998-05-15',
            .05, null, 'BONDS', .05, 1800.00, 8, 8, 'HSDIP',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2106, 120, 120, 15, 8, 0,0, '1997-04-30',
            .03, NULL, NULL, NULL, 1700.00, 0,0 ,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 2096, 160, 128, 15, 3, .04, 4500, '1999-02-18',
            .05, '1998-10-09', 'STOCK', .05, 2000.00, 0,0 ,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2096, 160, 160, 15, 3,0 ,0, '1998-02-01',
            .05, '1998-10-09', 'STOCK', .05, 2500.00,0 ,0 ,
            'HSDIP', null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2096, 120, 104, 15, 3,0 ,0, '1997-02-15',
            .06, NULL, NULL, NULL, 1700.00, 0,0 , 'HSDIP',
            null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2437, 120, 16, 15, 11.5,0 ,0, '1999-08-01',
            .035, NULL, NULL, NULL, 1800.00, 0,0 , 'GED',
            'MC655-6901', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2437, 120, 120, 15, 6.5,0 ,0, '1998-08-30',
            .03, NULL, NULL, NULL, 1200.00, 0,0 , 'GED',
            'MC655-6901', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2437, 120, 120, 15, 15, 0,0, '1997-08-16',
            .03, NULL, '401K', .05, 1100.00,0 ,0 , 'GED',
            'MC655-6901', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2598, 120, 120, 15, 15, 0,0, '1999-01-30',
            .035, NULL, NULL, NULL, 2150.50, 0,0 , 'HSDIP',
            'HP302-7409', 54.86);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2598, 120, 120, 15, 14, 0,0, '1998-01-15',
            .03, NULL, NULL, NULL, 1800.00,0 ,0 , 'HSDIP',
            'HP302-7409', 50.00);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2598, 120, 120, 15, 6, 0,0, '1997-02-01',
            .03, NULL, NULL, NULL, 1700.00, 0,0 , 'HSDIP',
            'HP302-7409', 45.75);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 3433, 120, 120, 15, 8,0 ,0, '1999-10-17',
            .05, NULL, NULL, NULL, 1400.00, 0, 0, 'JRCOLL',
            'MC655-7487', 84.05);

```

```

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3433, 120, 120, 15, 4, 0, 0, '1998-10-30',
    .05, NULL, NULL, NULL, 1300.00, 0 ,0 , 'JRCOLL',
    'MC655-7487', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3433, 120, 120, 15, 4, 0 ,0, '1997-10-15',
    .055, NULL, NULL, NULL, 1200.00,0 ,0 , 'JRCOLL',
    'MC655-7487', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 3778, 120, 120, 15, 0, 0 ,0, '1999-09-01',
    .055, NULL, NULL, NULL, 1240.50,0 ,0 , 'HSDIP',
    'HP302-7487', 54.86);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3778, 120, 120, 15, 14,0 ,0, '1998-09-26',
    .05, NULL, NULL, NULL, 1100.00, 0,0 , 'HSDIP',
    'HP302-7487', 50.00);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3778, 120, 120, 15, 10, 0, 0 ,0, '1997-09-18',
    .05, NULL, NULL, NULL, 1000.00,0 ,0 , 'HSDIP',
    'HP302-7487', 45.75);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 1034, 160, 112, 15, 6, .10, 5000, '1999-02-01',
    .05, NULL, 'BONDS', .06, 2850.60, 0, 0, 'HSDIP',
    'MC655-4490', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 1034, 160, 112, 15, 15, 0, 0 ,0, '1998-02-17',
    .05, NULL, 'BONDS', .06, 2720.80,0 ,0 , 'HSDIP',
    'MC655-4490', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 1034, 160, 48, 15, 8.5, 0,0 , '1997-02-15',
    .05, NULL, NULL, NULL, 2500.00, 0,0 , 'HSDIP',
    'MC655-4490', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2424, 120, 120, 15, 15,0 ,0, '1999-06-25',
    .04, NULL, NULL, NULL, 1900.00, 0,0 , 'HSDIP',
    'MC655-5571', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2424, 120, 120, 15, 7, 0, 0 ,0, '1998-07-01',
    .035, NULL, NULL, NULL, 1700.00,0 ,0 , 'HSDIP',
    'MC655-5571', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
    values (1997, 2424, 120, 120, 15, 3,0 ,0, '1997-07-17',
    .035, NULL, NULL, NULL, 1500.00,0 ,0 , 'HSDIP',
    'MC655-5571', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 2004, 160, 160, 15, 8, .04, 1550, '1999-02-17',
    .03, NULL, '401K', .04, 1850.00, 0 , 0 , 'JRCOLL',
    null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2004, 160, 160, 15, 2, 0,0 , '1998-02-01',
    .035, NULL, '401K', .04, 1700.00,0 ,0 , 'JRCOLL',
    null, null);

```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2004, 160, 160, 15, 3.5, 0,0, '1997-02-15',
       .03, NULL, NULL, NULL, 1600.00,0 ,0 , 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4456, 80, 80, 15, 3, 0,0, '1999-02-05',
       .03, NULL, NULL, NULL, 650.00,0 ,0 , 'HSDIP',
       'MC655-6680', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4456, 80, 80, 10, 0, 0,0, '1998-02-17',
       .02, NULL, NULL, NULL, 700.00,0 ,0 , 'HSDIP',
       'MC655-6680', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4456, 48, 48, 8, 1, 0 , 0 , null, null,
       NULL, NULL, NULL, 0,0 , 'HSDIP',
       'MC655-6680', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3288, 120, 120, 15, 9,0 ,0, '1999-02-01',
       .035, NULL, NULL, NULL, 1380.00,0 ,0 , 'HSDIP',
       'MC655-4402', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3288, 120, 120, 15, 8,0 ,0, '1998-02-03',
       .035, NULL, NULL, NULL, 1250.00,0 ,0 , 'HSDIP',
       'MC655-4402', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3288, 120, 120, 15, 11,0 ,0, '1997-01-28',
       .03, NULL, NULL, NULL, 1000.00, 0,0 , 'HSDIP',
       'MC655-4402', 70.00);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3341, 120, 120, 15, 9,0 ,0, '1999-07-25',
       .05, NULL, '401K', .06, 1350.00,0,0 , 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3341, 120, 116, 15, 8, 0,0, '1998-07-26',
       .06, null, '401K', .05, 1400.00, 16, 16, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3341, 120, 120, 15, 6.5, 0,0, '1997-07-15',
       .04, NULL, NULL, NULL, 900.00, 0,0 , 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2209, 120, 120, 15, 6, 0,0, '1999-07-02',
       .05, NULL, '401K', .05, 1200.00,0,0 , 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2209, 120, 120, 15, 7,0 ,0, '1998-06-17',
       .05, NULL, '401K', .05, 1200.00,0,0 , 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2209, 120, 120, 15, 3, 0,0, '1997-06-28',
       .045, null, null, null, 1550.80, 8, 8, 'COLL',
       null, null);
```

```

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3294, 120, 120, 15, 10, 0,0, '1999-02-20',
.05, NULL, '401K', .03, 1380.00,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3294, 120, 120, 15, 13, 0,0, '1998-01-28',
.05, NULL, '401K', .03, 1100.00,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3294, 120, 120, 15, 3, 0,0, '1997-02-04',
.05, NULL, '401K', .02, 1150.00,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3338, 120, 120, 15, 0, 0,0, '1999-07-17',
.05, NULL, NULL, NULL, 1200.00, 0,0 , 'HSDIP',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3338, 120, 120, 15, 1,0 ,0, '1998-07-19',
.045, NULL, NULL, NULL, 1130.00,0 ,0 , 'HSDIP',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3338, 120, 120, 15, 2, 0,0, '1997-07-08',
.05, NULL, NULL, NULL, 950.70, 0,0 , 'HSDIP',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2174, 160, 160, 15, 9, 0,0, '1999-09-26',
.055, NULL, '401K', .04, 1900.00,0 ,0 , 'JRCOLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2174, 160, 160, 15, 11, 0,0, '1998-09-10',
.05, NULL, '401K', .03, 1600.00, 0,0 , 'JRCOLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2174, 120, 120, 15, 8, 0,0, '1997-09-09',
.06, NULL, NULL, NULL, 1120.90, 0,0 , 'HSDIP',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3118, 120, 120, 15, 3, .05, 2000, '1999-11-02',
.04, NULL, 'BONDS', .08, 1350.60,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3118, 120, 112, 15, 8, 0,0, '1998-11-16',
.04, NULL, 'BONDS', .07, 1200.00,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3118, 120, 120, 15, 6,0 ,0, '1997-11-30',
.04, NULL, 'STOCK', .06, 1100.00,0,0, 'COLL',
null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3222, 120, 120, 15, 6, .04, 1780, '1999-01-16',
.06, '1999-06-08', '401K', .06, 1200.00, 32, 16, 'MAS',
null, null);

```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3222, 120, 120, 15, 4, 0, 0, '1998-01-28',
       .06, null, '401K', .06, 1150.00, 48, 8.5, 'MAS',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3222, 120, 120, 15, 7, 0, 0, '1997-01-13',
       .05, null, '401K', .05, 980.00, 16, 16, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4321, 120, 96, 15, 2, .05, 1720, '1999-08-24',
       .055, null, null, null, 1100.00, 16, 16, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4321, 80, 80, 15, 4, 0, 0, '1998-08-29',
       .05, NULL, NULL, NULL, 980.00, 0, 0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4321, 80, 80, 10, 4, 0, 0, '1997-08-08',
       .04, NULL, NULL, NULL, 850.00, 0, 0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2461, 120, 112, 15, 0, 0, 0, '1999-09-18',
       .05, NULL, NULL, NULL, 1950.00, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2461, 120, 120, 15, 4, 0, 0, '1998-09-01',
       .04, null, null, null, 1830.00, 48, 48, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2461, 120, 120, 15, 3, 0, 0, '1997-09-18',
       .035, NULL, NULL, NULL, 1600.00, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3841, 120, 120, 15, 1, 0, 0, '1999-10-05',
       .06, NULL, 'BONDS', .05, 1200.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3841, 120, 120, 15, 3, 0, 0, '1998-10-31',
       .05, NULL, 'BONDS', .05, 1020.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3841, 80, 80, 15, 2, 0, 0, '1997-10-11',
       .07, NULL, NULL, NULL, 980.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4002, 120, 120, 15, 3, 0, 0, '1999-12-01',
       .05, NULL, NULL, NULL, 1630.00, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4002, 120, 120, 15, 6, 0, 0, '1998-12-05',
       .04, NULL, NULL, NULL, 1400.00, 0, 0, 'HSDIP',
       null, null);
```

```

INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 4002, 80, 80, 15, 5, 0, 0, '1997-12-01',
        .04, NULL, NULL, NULL, 1380.00, 0, 0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 1003, 160, 56, 15, 0, .10, 11500, null,
        .05, NULL, '401K', .10, NULL, 0, 0, 'MAS',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1998, 1003, 160, 80, 15, 0, .10, 10000, null,
        .05, NULL, '401K', .10, NULL, 0, 0, 'MAS',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1997, 1003, 160, 40, 15, 0, .10, 10000, null,
        .05, NULL, '401K', .10, NULL, 0, 0, 'MAS',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 5103, 12, 12, 2, 0, 0, 0, null, null,
        NULL, NULL, NULL, 0, 0, 'HSDIP',
        'HP302-8403', 54.86);
INSERT INTO DEMOEMPL.BENEFITS
    values (1999, 2466, 120, 120, 15, 9, .05, 2650, '1999-10-26',
        .05, NULL, '401K', .03, 1800.00, 16, 16, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2466, 120, 112, 15, 11, 0, 0, '1998-10-18',
        .04, NULL, NULL, NULL, 1300.00, 0, 0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 2466, 120, 120, 15, 10, 0, 0, '1997-10-10',
        .035, NULL, NULL, NULL, 980.00, 0, 0, 'JRCOLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 3449, 120, 120, 15, 8, 0, 0, '1999-12-08',
        .04, NULL, NULL, NULL, 240.50, 0, 0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 3449, 120, 104, 15, 8, 0, 0, '1998-12-02',
        .05, NULL, NULL, NULL, 1100.00, 0, 0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1997, 3449, 120, 112, 15, 9, 0, 0, '1997-12-18',
        .03, NULL, NULL, NULL, 080.00, 0, 0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1999, 2781, 120, 120, 15, 8, 0, 0, '1999-04-11',
        .05, NULL, '401K', .03, 1700.00, 0, 0, 'COLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
    VALUES (1998, 2781, 120, 96, 15, 15, 0, 0, '1998-04-26',
        .05, NULL, '401K', .03, 1450.80, 0, 0, 'COLL',
        null, null);

```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2781, 120, 120, 15, 2, 0, 0, '1997-04-18',
       .05, NULL, NULL, NULL, 1100.00, 0, 0, 'COLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2894, 120, 48, 15, 1, 0, 0, '1999-05-01',
       .05, null, 'STOCK', .08, 1920.00, 16, 0, 'MAS',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2894, 120, 40, 15, 0, 0, 0, '1998-05-18',
       .08, null, 'STOCK', .08, 1750.00, 32, 32, 'MAS',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2894, 120, 0, 15, 0, 0, 0, '1997-05-11',
       .06, null, 'STOCK', .08, 1600.00, 16, 8, 'MAS',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3411, 120, 120, 15, 3, 0, 0, '1999-10-10',
       .04, NULL, '401K', .03, 1350.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3411, 120, 120, 15, 15, 0, 0, '1998-09-10',
       .04, NULL, '401K', .03, 1250.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3411, 120, 120, 15, 15, 0, 0, '1997-09-28',
       .03, NULL, NULL, NULL, 1100.00, 0, 0, 'JRCOLL',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (1999, 4358, 120, 112, 15, 2, .07, 1300, '1999-10-01',
       .055, NULL, NULL, NULL, 790.80, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (1998, 4358, 120, 80, 15, 0, .07, 1230, '1998-09-15',
       .055, NULL, NULL, NULL, 820.00, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (1997, 4358, 80, 80, 15, 14.5, .06, 980, '1997-09-26',
       .055, NULL, NULL, NULL, 700.00, 0, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 1234, 92, 40, 8, 12, .05, 9800, '2000-04-18',
       .06, '1998-07-10', 'BONDS', .10, 1750.00, 72, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (1999, 1234, 160, 16, 15, 0, .05, 8870, '1999-04-26',
       .07, '1998-07-10', 'BONDS', .08, 1600.00, 48, 0, 'HSDIP',
       null, null);
INSERT INTO DEMOEMPL.BENEFITS
values (1998, 1234, 160, 32, 15, 0, .05, 8440, '1998-04-10',
       .06, '1998-07-10', 'BONDS', .07, 1600.00, 56, 0, 'HSDIP',
       null, null);
```

```

INSERT INTO DEMOEMPL.BENEFITS
    values (1997, 1234, 160, 0, 15, 0, .05, 7690, '1997-04-01',
            .06, null, 'BONDS', .06, 1580.50, 48, 0, 'HSDIP',
            null, null);

INSERT INTO DEMOEMPL.INSURANCE_PLAN
    values ('PLI', 'Provident Life Insurance',
            '950 Gibraltar Ave', 'Lisbon', 'VA', '03097',
            '7033548300', 7815, null, 1000000, null, null, '1988-02-01');

INSERT INTO DEMOEMPL.INSURANCE_PLAN
    values ('HHM', 'Homostasis Health Maintenance Program',
            '57 Goodwill Blvd', 'Bellingham', 'MA', '01988',
            '5083535600', 2867, 300, 100000, 30, NULL, '1992-01-03');

INSERT INTO DEMOEMPL.INSURANCE_PLAN
    values ('HGH', 'Holistic Group Health Association',
            '2 Technology Park', 'Winnetka', 'IL', '06060',
            '9413865700', 9471, NULL, 900000, 10, 5, '1992-01-08');

INSERT INTO DEMOEMPL.INSURANCE_PLAN
    values ('DAS', 'Dental Associates',
            '52 Dedham Pl', 'Medford', 'MA', '03032',
            '6174445362', 5598, 50, 15000, NULL, NULL, '1993-01-04');

INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 2096, '1995-03-03',
        null, 1);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2096, '1995-03-03',
        null, 3);

INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 2096, '1995-03-03',
        null, 3);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2437, '1995-03-15',
        null, 2);

INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 2598, '1997-07-25',
        null, 1);

INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 3433, '1993-12-31',
        null, 1);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3433, '1993-11-01',
        '1993-12-31', 1);

INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3433, '1993-12-31',
        null, 1);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3778, '1998-01-21',
        NULL, 0);

INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3778, '1998-01-21',
        NULL, 0);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 1034, '1992-06-01',
        NULL, 0);

INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 1034, '1993-12-01',
        NULL, 0);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2424, '1993-07-24',
        NULL, 0);

INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 4456, '1994-01-04',
        null, 1);

INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3288, '1993-06-12',
        null, 1);

```

```
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3288, '1993-12-01',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3341, '1993-10-02',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3341, '1997-01-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 2209, '1992-08-12',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 2209, '1993-12-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3294, '1993-02-19',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3338, '1994-12-11',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 2299, '1996-01-01',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 3199, '1995-10-20',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3199, '1995-10-20',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3199, '1995-10-20',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 4001, '1995-12-11',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 4001, '1997-01-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 4008, '1996-01-23',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4008, '1996-01-23',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 4962, '1997-10-04',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4962, '1997-12-01',
null, 4);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3764, '1994-08-25',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 5090, '1998-07-12',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 4027, '1996-04-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3991, '1994-11-12',
'1995-12-31',5);
INSERT INTO DEMOEMPL.COVERAGE values ('HJM', 3991, '1996-01-01',
null, 5);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3991, '1994-11-12',
null, 5);
```

```
INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 1765, '1992-06-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 1765, '1993-12-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4773, '1995-10-14',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 3767, '1994-09-20',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3767, '1994-09-20',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3767, '1995-01-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 2448, '1992-01-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 2448, '1993-12-01',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3704, '1997-01-01',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HGH', 4703, '1997-03-19',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4703, '1997-03-19',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2246, '1992-06-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 2246, '1998-01-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 5008, '1998-01-31',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 5008, '1998-01-31',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 1234, '1993-06-01',
null, 5);
INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 2174, '1995-03-30',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 3118, '1995-07-23',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3222, '1995-10-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('PLI', 1003, '1988-02-01',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HHM', 1003, '1992-06-01',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 1003, '1993-12-01',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('PLI', 5103, '1999-10-11',
NULL, 0 );
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HHM', 5103, '1999-10-11',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 5103, '1999-10-11',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('PLI', 2781, '1995-09-27',
NULL, 0 );
```

```
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 2781, '1998-01-01',
null, 2);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('PLI', 2894, '1995-11-12',
null, 0 );
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HGH', 2894, '1995-11-12',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 2894, '1995-11-12',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HGH', 3411, '1997-01-30',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 3411, '1997-01-30',
null, 3);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('HHM', 4358, '1996-09-13',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE VALUES ('DAS', 4358, '1996-09-13',
null, 1);

INSERT INTO DEMOPROJ.ASSIGNMENT
VALUES (2466, 'D880', '1999-11-01', NULL);
INSERT INTO DEMOPROJ.ASSIGNMENT
values (2894, 'P634', '2000-02-15', null);
INSERT INTO DEMOPROJ.ASSIGNMENT
values (3411, 'P634', '2000-03-01', null);
INSERT INTO DEMOPROJ.ASSIGNMENT
VALUES (4358, 'C240', '1998-06-01', '1998-08-15') ;

UPDATE DEMOEMPL.DIVISION
SET DIV_HEAD_ID =2180
WHERE DIV_CODE = 'D02';
UPDATE DEMOEMPL.DIVISION
SET DIV_HEAD_ID =2010
WHERE DIV_CODE = 'D04';
UPDATE DEMOEMPL.DIVISION
SET DIV_HEAD_ID =4321
WHERE DIV_CODE = 'D06';
UPDATE DEMOEMPL.DIVISION
SET DIV_HEAD_ID =1003
WHERE DIV_CODE = 'D09';
UPDATE DEMOEMPL.DEPARTMENT
SET DEPT_HEAD_ID =3082
WHERE DEPT_ID = 3510 ;
UPDATE DEMOEMPL.DEPARTMENT
SET DEPT_HEAD_ID =2180
WHERE DEPT_ID = 2200 ;
UPDATE DEMOEMPL.DEPARTMENT
SET DEPT_HEAD_ID =2246
WHERE DEPT_ID = 1100 ;
UPDATE DEMOEMPL.DEPARTMENT
SET DEPT_HEAD_ID =3769
WHERE DEPT_ID = 3520 ;
UPDATE DEMOEMPL.DEPARTMENT
SET DEPT_HEAD_ID =2010
WHERE DEPT_ID = 2210 ;
```

```
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =1003
    WHERE DEPT_ID = 4200 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =1765
    WHERE DEPT_ID = 1110 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2004
    WHERE DEPT_ID = 1120 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2096
    WHERE DEPT_ID = 4600 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2209
    WHERE DEPT_ID = 3530 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2598
    WHERE DEPT_ID = 5100 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2461
    WHERE DEPT_ID = 6200 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2894
    WHERE DEPT_ID = 5200 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2466
    WHERE DEPT_ID = 5000 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =2466
    WHERE DEPT_ID = 4900 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =1003
    WHERE DEPT_ID = 6000 ;
UPDATE DEMOEMPL.DEPARTMENT
    SET DEPT_HEAD_ID =3222
    WHERE DEPT_ID = 4500 ;

COMMIT WORK RELEASE;
```



# Appendix C. Precompiler Directives

---

C.1 About this appendix . . . . .	C-3
C.2 Overriding DDLDML area ready mode . . . . .	C-4
C.2.1 Syntax . . . . .	C-4
C.2.2 Parameters . . . . .	C-4
C.3 No logging of program activity statistics . . . . .	C-5
C.3.1 Syntax . . . . .	C-5
C.3.2 Parameters . . . . .	C-5
C.4 Generating a source listing . . . . .	C-6
C.4.1 Syntax . . . . .	C-6
C.4.2 Parameters . . . . .	C-6
C.5 Usage . . . . .	C-7

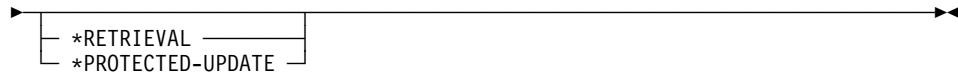


## C.1 About this appendix

This appendix provides information about CA-IDMS precompiler directives that are not associated with SQL statements and host variable declarations.

## C.2 Overriding DDLDML area ready mode

### C.2.1 Syntax



### C.2.2 Parameters

#### \*RETRIEVAL

Overrides the default ready mode for the DDLDML area of the dictionary by specifying that the area is to be readied for retrieval only. This allows concurrent database transactions to access the area in shared retrieval, shared update, protected retrieval, or protected update modes.

#### \*PROTECTED-UPDATE

Overrides the default ready mode for the DDLDML area of the dictionary by specifying that the area is to be readied for both retrieval and update. This allows concurrent database transactions to ready the area in shared retrieval mode only. The protected update usage mode prevents concurrent update of the area.

The dictionary ready override statement is printed on the source listing but is not passed to the COBOL compiler.

## C.3 No logging of program activity statistics

### C.3.1 Syntax

► ┌ \*NO-ACTIVITY-LOG ┐ ──────────────────────────────────►

### C.3.2 Parameters

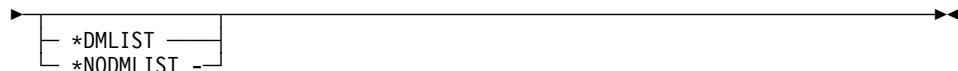
#### \*NO-ACTIVITY-LOG

Suppresses the logging of program activity statistics. The precompiler generates and logs the following program activity statistics unless the \*NO-ACTIVITY-LOG option is specified:

- Program name
- Language
- Date last compiled
- Number of lines
- Number of compilations
- Date created
- Schema name
- File statistics
- Database access statistics

## C.4 Generating a source listing

### C.4.1 Syntax



### C.4.2 Parameters

#### \*DMLIST

Specifies that the source listing is to be generated for the statements that follow.

\*DMLIST overrides a previous \*NODMLIST directive and the NOLIST precompiler parameter.

#### \*NODMLIST

Specifies that the source listing is not to be generated for the statements that follow.

\*NODMLIST overrides a previous \*DMLIST directive and the LIST precompiler parameter.

## C.5 Usage

*Column position:* Precompiler directives must be coded beginning in column 7.

*Default ready mode:* The default ready for the DDLDML area mode is shared update. Shared update readies the area for both retrieval and update and allows concurrent database transactions to ready the DDLDML area in shared update or shared retrieval.

*Program activity statistics:* Program activity statistics will not be logged if the DDLDML area is readied for retrieval only.



## **Appendix D. Calls to IDMSIN01**

---

D.1	About this appendix . . . . .	D-3
D.2	How to call IDMSIN01 . . . . .	D-4
D.3	Example . . . . .	D-5



## D.1 About this appendix

This appendix provides examples of calls to IDMSIN01. IDMSIN01 is an entry point to the IDMS module that provides various IDMS functions to user programs, including:

- Deactivate the SQL trace
- Reactivate the SQL trace
- Retrieve (GETPROF) user profile information
- Establish (SETPROF) user profile information
- Retrieve SQL error messages into a user buffer
- Translate an internal 8-byte DATETIME stamp to displayable form
- Return the current DATE and TIME in a displayable form
- Translate a displayable 26-byte DATETIME value to an eight-byte DATETIME stamp

## D.2 How to call IDMSIN01

You use standard calling conventions to call IDMSIN01. The first two parameters passed are the address of an RPB block and the address of the function REQUEST-CODE and RETURN-CODE fields.

**If the language is ADS:** If the language is ADS, issue calls to IDMSIN01 as shown below:

```
LINK TO 'IDMSIN01' USING (SQLRPB, REQ-WK, SQLCA, SQLMSG).
```

## D.3 Example

Calls to IDMSIN01 for the functions listed above are shown in the example below.  
The example uses COBOL calling conventions.

```

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.

01 SQLRPB.
  02 FILLER          PIC X(36).

01 SQLMSGS.
  02 SQLMMAX        PIC S9(8) COMP VALUE +6.
  02 SQLMSIZE        PIC S9(8) COMP VALUE +80.
  02 SQLMCNT         PIC S9(8) COMP.
  02 SQLMLINE        OCCURS 6 TIMES PIC X(80).

01 REQ-WK.
  02 REQUEST-CODE    PIC S9(8) COMP.
  02 REQUEST-RETURN   PIC S9(8) COMP.

01 WORK-FIELDS.
  02 WK-DTS-FORMAT   PIC S9(8) COMP VALUE 0.
  02 LINE-CNT         PIC S9(4) COMP.
  02 SQLVALUE         PIC ----9.
  02 WK-CDTS          PIC X(26).
  02 WK-KEYWD         PIC X(8).
  02 WK-VALUE          PIC X(32).
  02 WK-DBNAME         PIC X(8).

EXEC SQL
  INCLUDE SQLDA
END-EXEC.
EXEC SQL  BEGIN DECLARE SECTION      END-EXEC
01 SQLID           USAGE SQLSESS.
01 ONEROW          PIC S9(4) COMP VALUE +1.
01 WK-ROWS.
  02 WK-ROW OCCURS 80 TIMES.
  03 WK-BYTE PIC X.
EXEC SQL  END  DECLARE SECTION      END-EXEC.
01 WK-DATA REDEFINES WK-ROWS.
  02 WK-SCHEMA        PIC X(18).
  02 WK-CUSER         PIC X(18).
  02 WK-DTS          PIC X(8).

*****
PROCEDURE DIVISION.
*****



*****
* Call IDMSIN01 to deactivate the SQL trace
* which was originally activated by the
* SYSIDMS parm SQLTRACE=ON.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*****

```

```
MOVE 1 TO REQUEST-CODE
CALL 'IDMSIN01' USING SQLRPB REQ-WK.

*****
* Call IDMSIN01 to reactivate the SQL trace
* which was originally activated by the
* SYSIDMS parm SQLTRACE=ON and
* has been previously deactivated earlier on in this job.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*****
MOVE 0 TO REQUEST-CODE
CALL 'IDMSIN01' USING SQLRPB REQ-WK.
```

### D.3 Example

---

```
*****
* Call IDMSIN01 to request a 'GETPROF' to get the user
* profile default DBNAME, which was established by the
* SYSIDMS parm DBNAME=xxxxxxxx when running
* in local mode, or by DCUF SET DBNAME xxxxxxxx
* when running under central version.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the 8-byte GETPROF keyword.
* Parm 4 is the address of the 32-byte GETPROF returned value.
*****

MOVE 2 TO REQUEST-CODE
MOVE 'DBNAME' TO WK-KEYWD
CALL 'IDMSIN01' USING SQLRPB REQ-WK WK-KEYWD
WK-VALUE.
MOVE WK-VALUE TO WK-DBNAME.
IF WK-DBNAME = SPACES
  DISPLAY 'DBNAME is set to BLANKS'
  ELSE
    DISPLAY 'DBNAME is set to ' WK-DBNAME.

*****
* Call IDMSIN01 to request a 'SETPROF' to set the user
* profile default SCHEMA to the value 'SYSTEM'.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the 8 byte SETPROF keyword.
* Parm 4 is the address of the 32 byte SETPROF value.
*****

MOVE 3 TO REQUEST-CODE
MOVE 'SCHEMA' TO WK-KEYWD
MOVE 'SYSTEM' TO WK-VALUE
CALL 'IDMSIN01' USING SQLRPB REQ-WK WK-KEYWD
WK-VALUE.
IF REQUEST-RETURN NOT = 0
  MOVE REQUEST-RETURN TO SQLVALUE
  DISPLAY 'SETPROF returned error ' SQLVALUE.
```

```
*****
* Call IDMSIN01 to retrieve SQL error messages, associated
* with an SQLCODE value less than 0, into a user buffer
* that will then be displayed back to the user. What is
* passed is the SQLCA block and a message control block
* consisting of the following fields:
*
*   - Maximum number of lines in user buffer
*   - The size (width) of one line in the user buffer
*   - The actual number of lines returned from IDMSIN01
*   - The user buffer where the message lines are returned
*
* A return code of 4 means that there were no SQL error messages.
* A return code of 8 means that there were more SQL error lines
* in the SQLCA than could fit into the user buffer, meaning
* truncation has occurred.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the SQLCA block.
* Parm 4 is the address of the message control block.
*****
MOVE 4 TO REQUEST-CODE.
CALL 'IDMSIN01' USING SQLRPB, REQ-WK,
      SQLCA, SQLMSGS.
IF REQUEST-RETURN NOT = 4
  MOVE 1 TO LINE-CNT
  PERFORM DISP-MSG UNTIL LINE-CNT > SQLMCNT.

DISP-MSG SECTION.
  DISPLAY SQLMLINE (LINE-CNT).
  ADD 1 TO LINE-CNT.
```

```
*****
* Call IDMSIN01 to have an 8 byte internal DATETIME stamp
* returned as a displayable 26 character DATE/TIME display.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the 4 byte format indicator (0's).
* Parm 4 is the address of the 8 byte internal DATETIME stamp.
* Parm 5 is the address of the 26 byte DATE/TIME returned.
*****
MOVE 5 TO REQUEST-CODE
MOVE 'UNKNOWN' TO WK-CDTS
MOVE 0 TO WK-DTS-FORMAT
CALL 'IDMSIN01' USING SQLRPB REQ-WK
    WK-DTS-FORMAT WK-DTS WK-CDTS.
DISPLAY 'THE DATE AND TIME IS --> ' WK-CDTS.

*****
* Call IDMSIN01 to have the current DATE and TIME
* returned as a displayable 26 character DATE/TIME display.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the 4 byte format indicator (1).
* Parm 4 is the address of the 26 byte DATE/TIME returned.
*****
MOVE 5 TO REQUEST-CODE
MOVE 1 TO WK-DTS-FORMAT
CALL 'IDMSIN01' USING SQLRPB REQ-WK
    WK-DTS-FORMAT WK-CDTS
DISPLAY 'THE DATE AND TIME IS --> ' WK-CDTS.

*****
* Call IDMSIN01 to have a 26 byte external DATE/TIME display
* returned as an 8 byte DATETIME stamp.
*
* Parm 1 is the address of the RPB.
* Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
* Parm 3 is the address of the 4 byte format indicator (2).
* Parm 4 is the address of the 26 byte DATE/TIME.
* Parm 5 is the address of the 8 byte DATETIME stamp returned.
*****
MOVE 5 TO REQUEST-CODE
MOVE 2 TO WK-DTS-FORMAT
MOVE '1994-07-18-12.01.18.458382' TO WK-CDTS
CALL 'IDMSIN01' USING SQLRPB REQ-WK
    WK-DTS-FORMAT WK-CDTS WK-DTS.
```

**If the language is ADS:** To extract messages from the SQLCA in ADS, perform the following steps:

1. Replace the PERFORM statement with a loop that moves SQLMLINE to an error field for the count specified in SQLMCNT
2. Display the messages as follows:

```
DISPLAY MESSAGE TEXT IS error-field
```

### D.3 Example

---

# Index

---

## A

access module  
    authority to use 5-11  
    automatic re-creation 1-12, 5-12  
    BS2000/OSD JCL to create A-19  
    changing 5-15  
    default isolation for 5-13  
    defaults 5-11, 6-7  
    definition 5-11  
    execution at runtime 5-17  
    how to execute a test version 5-17  
    OS/390 JCL to create A-4  
    precompiler specification 5-6  
    schema-name mapping 5-12, 5-15  
    SET ACCESS MODULE statement 6-7  
    timestamp validation 5-12  
    transaction state for 5-13  
    version 5-12  
VM/ESA commands to create A-14  
VSE/ESA JCL to create A-10  
ALTER ACCESS MODULE statement 5-15

## B

BEGIN DECLARE SECTION 2-5  
END DECLARE SECTION 2-5  
bill-of-materials explosion with SQL 6-21  
bulk buffer 2-6  
bulk fetch 3-19—3-22  
    checking statement status 3-20  
    example with dynamic SQL 7-10  
    for scrolling through rows 6-10  
ROWS parameter 3-19  
START parameter 3-20  
bulk insert 3-23—3-24  
    ROWS parameter 3-23  
    START parameter 3-23  
bulk processing 3-19—3-24  
    data type of, indicator variable 2-4  
    defined 1-5  
bulk select 3-22—3-23  
bulk structure  
    in CA-ADS 4-9—4-10  
    in COBOL 4-16, 4-19—4-20  
    in PL/I 4-34

## C

CA-ADS applications, embedding SQL  
continuing statements 4-4  
declaration module 4-6  
declaring host variables 4-6  
delimiters 4-4  
equivalent data types 4-6  
including a table 4-8  
including SQLCA 4-10  
inserting comments 4-5  
order of dialog compilation 4-5  
placing statements 4-5  
qualifying host variable names 4-10  
requirements 4-4  
scope of DECLARE CURSOR 4-6  
scope of WHENEVER 4-6  
SQLCA structure 4-11  
CA-IDMS SQL extensions  
    *See* SQL extensions  
CA-OLQ 5-18  
CALL  
    procedure 1-5  
CALL Statement  
cardinality violation 3-5, 3-22  
central version 1-13, 5-13, 5-16  
check constraint 1-6  
CICS, effect of statements on processing 2-11  
COBOL applications, embedding SQL  
    COBOL version considerations 4-14  
    continuing statements 4-12  
    declaring host variables 4-14, 4-26  
    declaring SQLCA 4-23  
    delimiters 4-12  
    equivalent data types 4-15  
INCLUDE TABLE statement 4-17  
indicator variables 4-16  
inserting comments 4-13  
placing statements 4-13  
qualifying host variable names 4-22  
requirements 4-12  
SQLCA structure 4-23  
    subscripted host variable names 4-23  
COBOL applications, precompiling 5-7  
column list, in INSERT 3-6  
Command Facility, in debugging 5-18  
commands  
    VM/ESA A-14

---

committing data  
    COMMIT statement 2-10  
    without ending the transaction 2-10

compiling 1-11, 5-10

concurrent access to an area 2-14

concurrent processing 6-9

concurrent sessions 6-15—6-17  
    session identifier 6-15, 6-16  
    SQLSESS host variable 6-15  
    steps to manage 6-16

constraint violation  
    on DELETE 3-9  
    on INSERT 3-7

COPY IDMS FILE 4-25

COPY IDMS MODULE 4-27

COPY IDMS RECORD 4-25

CREATE ACCESS MODULE statement  
    AUTO RECREATE option 5-12  
    DEFAULT ISOLATION parameter 5-13  
    INCREMENTAL option of READY parameter 5-13  
    MAP schema parameter 5-12  
    PRECLAIM option of READY parameter 5-13  
    READ ONLY transaction state 5-13  
    READ WRITE transaction state 5-13  
    VALIDATE BY option 5-12

CREATE TEMPORARY TABLE statement 6-18

creating an access module 1-11

cursor  
    closing 3-14  
    cursor position 1-4, 3-19  
    declaring 3-12  
    defined 1-4  
    external 6-4  
    fetching from 3-13  
    for temporary tables 6-19  
    global 6-4  
    in bill-of-materials explosion 6-21  
    invalid cursor state 3-14, 3-16  
    no more rows 3-13  
    opening 3-13  
    position 3-13  
    shared 6-4  
    updateable 1-4, 3-12  
    using 3-12—3-22

cursor stability 6-11, 6-12

data manipulation (*continued*)  
    bill-of-materials explosion 6-21  
    bulk processing 3-19—3-24  
    checking for modified rows after a  
        pseudoconverse 6-13  
    checking statement status 3-9, 3-20, 3-24  
    DELETE statement 3-9  
    deleting all rows of a table 3-10  
    deleting data 3-17—3-18  
    FETCH statement 3-13, 3-19  
    INSERT statement 3-6, 3-23  
    modifying data 3-7—3-9, 3-10, 3-15  
    positioned delete 3-17—3-18  
    positioned update 3-15—3-17  
    retrieval 3-4—3-5  
    retrieving data 3-10, 3-13, 3-17, 3-19—3-23  
    ROWS parameter, on FETCH 3-19  
    ROWS parameter, on INSERT 3-23  
    searched delete 3-9—3-10  
    searched update 3-8—3-9, 6-12  
    SELECT statement 1-4, 3-4, 3-22  
    SET ACCESS MODULE statement 5-17  
    SQL DML operations 3-4  
    START parameter, on FETCH 3-20  
    START parameter, on INSERT 3-23  
    UPDATE statement 3-7  
    updateable cursor 3-15  
    updating after a pseudoconverse 6-9  
    using a bulk fetch 3-19—3-22  
    using a bulk insert 3-23—3-24  
    using a bulk select 3-22—3-23  
    using a cursor 3-12—3-22  
        with null values 3-10—3-11

database transaction  
    beginning and ending 2-10  
    continuing 2-10  
    definition 2-10  
    effect of teleprocessing statements on 2-11

database, demonstration B-14

database, test  
    table descriptions B-4  
    test data B-10

date format 5-8

DATETIME stamp, translating D-10

DC task-level statements 2-11

debugging 5-18—5-20

declaration module 4-5, 4-6

declaring a global cursor 6-4

declaring an external cursor  
    requirements 6-4  
    user validation 6-5

## D

data exception error, on INSERT 3-7  
data manipulation  
    adding data 3-6—3-7, 3-10, 3-23—3-24

---

deleting data 3-9  
demonstration database B-14  
dictionary 4-25  
dynamic SQL  
    checking statement status 7-7, 7-9  
    limited by no host variables 7-4  
    programs with only dynamic SQL 7-4  
    requirements 7-4  
    update operations 7-6  
    when to use EXECUTE 7-9  
    when to use EXECUTE IMMEDIATE 7-6  
    when to use PREPARE 7-7

## E

embedded SQL  
    in CA-ADS applications 4-4—4-11  
    in COBOL applications 4-12—4-28  
    in PL/I applications 4-29—4-40  
    programming functions 1-10  
error messages, displaying D-8  
errors, SQL  
    error codes 2-19  
    error message, displaying 2-21  
    error-handling techniques 2-21—2-22  
    SQLCODE error values 2-18  
    syntax error in prepared statement 7-8  
EUR date/time format 5-8  
executing an SQL program 1-12, 5-16  
EXPLAIN statement 5-19  
External routines 3-25—3-27

## F

FETCH statement 1-4

## G

GETPROF D-8

## H

host variable  
    defined 1-4  
    definition 2-4—2-6, 4-16, 4-30—4-35  
    reference requirements 2-7  
    references to in COBOL 4-22  
    references to in PL/I 4-35  
    SQLSESS 6-15  
    to dynamically specify access module 6-8  
host variable array 3-19

I  
IDD 4-25  
IDMSCINT 2-11  
IDMSIN01 entry point 2-21, 5-19  
    calls to D-3—D-11  
    example D-4—D-11  
GETPROF D-8  
getting a profile with D-8  
SETPROF D-8  
setting a profile with D-8  
SQL trace facility D-6  
translating internal DATETIME stamp with D-10  
using ADS D-4, D-10  
INCLUDE IDMS module statement 4-39  
INCLUDE IDMS record statement 4-38  
INCLUDE module 4-27, 4-40  
INCLUDE TABLE  
    authorization requirements 5-5  
    determining schema qualifier 5-6  
    for declaring host variables 2-5  
    guidelines 2-7  
    options 2-6  
indicator array  
    in COBOL 4-20—4-22  
indicator variable  
    data type of 2-4  
    definition 2-4  
    SQLIND data type 2-4, 4-16, 4-32  
    using 3-10—3-11  
integrity constraints  
    check constraint 1-6  
    constraint violation 1-7  
    data type 1-6  
    described 1-5—1-7  
    domain constraints 1-6  
    null constraint 1-6  
    referential constraint 1-5  
    unique constraint 1-5  
internal DATETIME stamp, translating D-10  
invalid SQL statement identifier error 5-6  
Invoking external routines  
ISO date/time format 5-8  
isolation level  
    concurrency control 2-13—2-14  
CREATE ACCESS MODULE statement 2-14  
SET TRANSACTION statement 2-14  
specified for access module 5-13  
specifying 2-14  
types 2-13

## J

### JCL

BS2000/OSD A-19  
OS/390 A-4  
VSE/ESA A-10  
JIS date/time format 5-9

## L

link editing 5-10  
local mode 1-13, 5-13, 5-16  
locks  
    during a suspended session 6-11  
    for a positioned update 3-15  
    for single-row select 3-5  
management 1-13  
types 2-13

## M

modularized programming 6-4  
multiple sessions  
    started by different programs 6-17  
    started by one program 6-16  
multiple-row insert 3-6, 3-7  
multiple-row select 3-5

## N

non-bulk structure  
    in COBOL 4-20  
non-SQL defined databases  
    accessing 1-8—1-9  
null value  
    definition 2-4, 2-5  
    testing for 3-10  
nxn-bulk structure  
    in COBOL 4-22

## O

online debugger 5-20  
OPEN statement 6-4  
optimizer 1-11, 5-11, 5-19

## P

paging application 6-10, 6-11  
PL/I applications, embedding SQL  
    continuing statements 4-30  
    copying dictionary source 4-37

### PL/I applications, embedding SQL (*continued*)

    data types of included table 4-32  
    declaring host variables 4-30, 4-38  
    declaring SQLCA 4-36  
    delimiters 4-29  
    equivalent data types 4-30  
    indicator variables 4-32  
    inserting comments 4-30  
    qualifying host variable names 4-35  
    requirements 4-29  
    SQLCA structure 4-36  
    subscripted host variable names 4-35  
    using INCLUDE TABLE 4-32

PL/I standard modules 4-39

precompiler  
    authorization requirements 5-5  
    BS2000/OSD JCL A-19  
    COBOL-specific options 5-7  
    functions 5-4  
    messages, with LIST option 5-7  
    options in JCL 5-5  
    OS/390 JCL A-4  
    output 5-10  
    SQL standards enforcement 5-7  
    VM/ESA commands A-14  
    VSE/ESA JCL A-10

precompiler directives 4-28, 4-40, C-3—C-7

precompiler-directive statement

    COPY IDMS FILE (COBOL) 4-25  
    COPY IDMS MODULE (COBOL) 4-27  
    COPY IDMS RECORD (COBOL) 4-25  
    INCLUDE IDMS module (PL/I) 4-39  
    INCLUDE IDMS record (PL/I) 4-38  
    INCLUDE module (COBOL) 4-27  
    INCLUDE module (PL/I) 4-40  
    INCLUDE TABLE (COBOL) 4-17  
    INCLUDE TABLE (PL/I) 4-32

precompiling 1-10, 5-4—5-9

prepared statement 2-10, 7-4

    defined 1-5

primary key  
    defined 1-5  
    specified in the search condition 3-5, 6-13

profile

    getting, with call to IDMSIN01 D-8  
    setting, with call to IDMSIN01 D-8

program activity statistics

*See* log suppression

pseudoconversational programming 6-9—6-14  
    checking for modified rows 6-13  
    definition 6-9

---

pseudoconversational programming (*continued*)  
  searched update in 6-12

## R

RCM 1-10, 5-4  
  dropping from an access module 5-15  
NOINSTALL precompiler option 5-6  
precompiler parameter 5-6  
replacing in an access module 5-15  
version, specified to precompiler 5-6  
ready mode 2-14  
  access module specification 5-13  
  actual ready mode 5-14  
  default 5-13  
  depending on transaction state 5-14  
repeatable reads of data 2-14  
rollback, automatic  
  on searched delete 3-9  
  when searched update fails 3-9  
row lock 2-13  
runtime processing of SQL statements 1-12

## S

sample program  
  bill-of-materials explosion 6-24  
  executing a prepared SELECT 7-10  
schema  
  defined 1-4  
  precompiler specification 5-6  
security  
  as applied to SQL access 5-16  
  CA-IDMS internal security 5-16  
  executing access modules 5-11  
  external security 5-16  
  role of schema ownership 5-16  
SELECT Statement  
SETPROF D-8  
single-row INSERT 3-6  
single-row select 3-4  
SQL access, terminology of 1-4—1-7  
SQL applications  
  application development steps 1-10  
  compilation steps 1-11  
  debugging 1-13  
  execution environments 1-13  
  testing 1-13  
SQL Communication Areas  
  error message, displaying 2-21  
  field values, displaying 2-20

SQL Communication Areas (*continued*)  
  SQLCA 2-15  
  SQLPIB 2-20  
  SQLSTATE 2-15  
SQL Communications Areas  
  including 4-10  
SQL DDL  
  for demonstration database B-14  
SQL declare section 2-5, 4-14, 4-16, 4-23, 4-27, 4-30, 4-31  
SQL extensions  
  bulk processing 1-5, 3-19  
  COBOL data structures 4-17  
  committing updates 2-10  
  data types 4-31  
  dynamic SQL 1-5  
  PL/I host variable definitions 4-32  
  session management 2-9  
SQL messages, displaying 2-21  
SQL session  
  automatic termination 2-9  
  beginning and ending 2-8—2-9  
  concurrent sessions 2-9  
  definition 2-8  
  effect of teleprocessing statements on 2-11  
RESUME statement 2-9  
SUSPEND statement 2-9  
SQL standards 5-7  
SQL trace facility 5-19, D-6  
SQLCA  
  description 2-15  
  fields 2-20  
  initialization 2-20  
  SQLCERC 2-19  
  SQLCODE 2-18  
SQLCNRP  
  checking on a bulk insert 3-24  
  checking on a bulk select 3-22  
  testing for bulk fetch 3-20  
SQLCODE 2-18, 2-20, 3-5, 3-7, 3-8, 3-9, 3-13, 3-20, 3-22, 3-24, 4-23, 4-36, 7-7  
SQLDA  
  checking statement status 7-8  
  declaring 7-7  
  declaring in CA-ADS 7-7  
  structure 7-7  
  values 7-8  
SQLPIB 2-20  
SQLSTATE  
  ANSI-defined values 2-15  
  CA-IDMS-defined values 2-15

---

**SQLSTATE** (*continued*)

ISO-defined values 2-15  
suppress log  
    *See* log suppression  
SYNCPOINT (CICS) statement 2-11  
syntax  
    for COPY IDMS FILE 4-25  
    for COPY IDMS module 4-27  
    for COPY IDMS RECORD 4-26  
    for INCLUDE IDMS module 4-39  
    for INCLUDE IDMS record 4-38  
    for precompiler directives C-4—C-6  
    for precompiler options 5-5  
    for SQLXQ1 ENTRY 4-29  
SYSIDMS parameters 2-8, 5-6, 5-16  
SQLTRACE= 5-19

**T**

table

adding data to 3-6  
base table 3-12  
constraints on values 1-5  
defined 1-4  
defining to a CA-ADS dialog 4-6  
deleting data from 3-9  
if a column is added 3-6  
including the definition in a program 2-5  
modifying data in 3-7  
name qualifier 5-12  
primary key 6-13  
result table 1-4, 3-4, 3-12, 3-19, 6-23, 7-4, 7-8  
row lock 2-13  
selecting data from 3-4  
timestamp column for 6-13  
updating through a cursor 1-4, 3-12  
table procedure 1-8  
teleprocessing statements 2-11  
temporary table 6-18—6-20  
    defined 1-5  
    differences from base tables 6-18  
    dropping, automatic 2-10  
    naming considerations 6-18  
    uses 6-18  
test versions 5-17  
time format 5-8  
timestamp column for a table 6-13  
transaction state 5-13, 5-14

**U**

USA date/time format 5-8

**V**

view

cannot use temporary table 6-18  
defined 1-4  
including the definition in a program 2-5  
name qualifier 5-12  
selecting data through 3-4  
updateable view 3-12





