CA IDMS - 19.0
Programming IDMS SQL

Date: 15-Jan-2018
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or 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 YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
# Table of Contents

**Accessing Data Using SQL** ................................................................. 15
SQL Database Access ........................................................................ 15
  Integrity Constraints ....................................................................... 17
Network-defined Database Access ....................................................... 19

**SQL Application Development** ..................................................... 19
Writing the Application ...................................................................... 20
Creating Executable Modules ............................................................. 20
Executing the Application .................................................................. 23
Testing and Debugging the Application .............................................. 23

**Writing an SQL Program** ............................................................... 24
Host Variables in an SQL Statement .................................................... 25
  How Host Variables Are Used ....................................................... 25
  Host Variable Example ................................................................ 25
  Indicator Variable ...................................................................... 25
  Null Value .................................................................................. 26
SQL Declare Sections ........................................................................ 26
  What You Can Do ...................................................................... 26
  Host Variable Declaration Example .......................................... 26
INCLUDE TABLE Directive ............................................................... 27
  INCLUDE TABLE Statement ....................................................... 27
  Statement Example .................................................................. 27
  Structure Example .................................................................. 27
  INCLUDE Statement Options .................................................... 27
  Including an Array .................................................................... 27
  Host Variable Array Structure .................................................. 28
  Usefulness of INCLUDE TABLE ................................................ 28
  When Not to Use INCLUDE TABLE ........................................... 28
Referring to Host Variables .............................................................. 29
  Reference Requirements .......................................................... 29
  Reference Example .................................................................. 29
Local Variables and Routine Parameters .......................................... 29
  How Local Variables and Routine Parameters Are Used ............. 29
Using a Cursor

Declaring a Cursor

How You Declare a Cursor

Updateable Cursors

Using Indicator Variables in Data Manipulation

Indicator Variables in SELECT or FETCH

Indicator Variables in Inserts and Updates

Update Examples With Indicator Variables

Using a Cursor

Declaring a Cursor

How You Declare a Cursor

Updateable Cursors

Data Manipulation with SQL

SQL DML Statements

Retrieving Data

Using SELECT

Single-row SELECT Statement

Checking Single-row Select Status

Updating the Single Row

Multiple-row SELECT

Adding Data

Using INSERT

Single-row INSERT

Guidelines for INSERT

Inserting Multiple Rows

Modifying Data

Using UPDATE

Checking UPDATE Status

Searched Updates

Searched Updates Using Host Variables

Searched Updates Without Host Variables

No Matching Rows

Automatic Rollback

Deleting Data

Using DELETE

Searched Deletes

Automatic Rollback

Using Indicator Variables in Data Manipulation

Overriding WHENEVER

Checking Specific Errors

When To Do It

How to Do It

Using WHENEVER SQLERROR

Using GET DIAGNOSTICS

Programming IDMS SQL 6
Updateable Cursor Declaration Example .............................................................. 62
Fetching a Row ........................................................................................................ 62
  Opening the Cursor .............................................................................................. 62
  How You Fetch a Row ......................................................................................... 62
  Closing a Cursor .................................................................................................. 64
  Automatic Closing of a Cursor ............................................................................ 64
  Invalid Cursor State ............................................................................................ 64
Executing a Positioned Update or Delete ............................................................... 65
  Requirements for a Positioned Update ............................................................... 65
  Advantage of an Updateable Cursor ................................................................. 65
  Positioned Update Example ............................................................................... 66
  Positioned Deletes ............................................................................................... 66
  Checking Positioned Delete Status ..................................................................... 67
  Positioned Delete Example ................................................................................ 67
Bulk Processing .................................................................................................... 67
Executing a Bulk Fetch ......................................................................................... 68
  Cursor Position ................................................................................................... 68
  How Many Rows Are Fetched? ............................................................................ 68
  Maximum Rows Example ................................................................................... 69
  Specifying a Starting Row .................................................................................. 69
  Checking Statement Execution .......................................................................... 69
  SQLCODE and SQLCNRP Values ..................................................................... 70
  Advantages of a Bulk Fetch ................................................................................ 70
  Bulk Fetch Considerations ................................................................................ 70
  Bulk Fetch Example ............................................................................................ 70
Executing a Bulk Select ......................................................................................... 71
  Checking the Status of a Bulk Select ................................................................. 71
  Advantage of a Bulk Select ................................................................................ 72
  Bulk Select Considerations ................................................................................ 72
Executing a Bulk Insert ........................................................................................ 72
  Specifying the START and ROWS Parameters ................................................ 72
  Bulk Insert Example ........................................................................................... 73
  Checking Bulk Insert Status ............................................................................... 73
  Advantage of a Bulk Insert ................................................................................ 73
Invoking Procedures ............................................................................................. 73
CALL Statement .................................................................................................. 74
  CALL of a Procedure .......................................................................................... 74
  CALL of a Table Procedure ............................................................................... 74
  Declaration of the Cursor .................................................................................. 74
  Opening the Cursor ............................................................................................ 74
  Fetching the Result Sets ...................................................................................... 75
SELECT Statement ................................................................................................. 75
Using SQL in a PL/I Application Program ................................................................. 102
   Embedding SQL Statements ................................................................................. 103
   Declaring SQLXQ1 ............................................................................................ 103
   Delimited, Continued, and Commented Statements .............................................. 103
   Defining Host Variables ..................................................................................... 104
   Using PL/I Declarations .................................................................................... 105
   Using INCLUDE TABLE ...................................................................................... 106
   Defining Bulk Structures .................................................................................... 108
   Referring to Host Variables ................................................................................ 109
   Including SQL Communication Areas ................................................................. 110
   Including Information from the Dictionary .......................................................... 111
   INCLUDE IDMS Record Statement .................................................................... 111
      Syntax .............................................................................................................. 111
      Parameters ..................................................................................................... 112
      Usage .............................................................................................................. 112
   INCLUDE IDMS MODULE statement .................................................................. 112
      Syntax .............................................................................................................. 112
      Parameters ..................................................................................................... 113
   INCLUDE Module-name Statement ..................................................................... 113
   Non-SQL Precompiler Directives ........................................................................ 114

Preparing and Executing the Program ................................................................. 114
   Precompiling the Program .................................................................................. 114
      About the Precompiler .................................................................................... 115
      Why You Precompile ..................................................................................... 115
      When to Precompile ....................................................................................... 115
      How You Precompile ..................................................................................... 115
      Authorization .................................................................................................. 115
      Precompiler Options ....................................................................................... 116
      Syntax .............................................................................................................. 116
      Parameters ..................................................................................................... 116
   Compiling the Program ....................................................................................... 119
      CA IDMS Precompiler ..................................................................................... 119
      Language Compiler ........................................................................................ 120
      Link Editing .................................................................................................... 120
   Creating the Access Module ............................................................................... 120
      How You Create an Access Module ................................................................. 121
      Overriding Access Module Defaults ................................................................ 121
         Access Module Name Qualifier ................................................................... 121
         Access Module Version Number ................................................................ 122
         Schema-name Mapping for Tables and Views ................................................. 122
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Access Module Re-creation</td>
<td>122</td>
</tr>
<tr>
<td>Table Definition Timestamp Validation</td>
<td>123</td>
</tr>
<tr>
<td>Transaction State</td>
<td>123</td>
</tr>
<tr>
<td>Isolation Level</td>
<td>123</td>
</tr>
<tr>
<td>Ready Mode</td>
<td>123</td>
</tr>
<tr>
<td>Default Ready Mode</td>
<td>124</td>
</tr>
<tr>
<td>Actual Ready Mode</td>
<td>124</td>
</tr>
<tr>
<td>READ ONLY Ready Modes</td>
<td>124</td>
</tr>
<tr>
<td>READ WRITE Ready Modes</td>
<td>124</td>
</tr>
<tr>
<td>Altering an Access Module</td>
<td>125</td>
</tr>
<tr>
<td>What You Can Change</td>
<td>125</td>
</tr>
<tr>
<td>Changing Schema-name Mapping</td>
<td>125</td>
</tr>
<tr>
<td>Executing an SQL Application</td>
<td>125</td>
</tr>
<tr>
<td>Batch Jobs</td>
<td>125</td>
</tr>
<tr>
<td>SYSIDMS Parameters</td>
<td>125</td>
</tr>
<tr>
<td>Execution Privilege</td>
<td>126</td>
</tr>
<tr>
<td>Testing the Access Module</td>
<td>127</td>
</tr>
<tr>
<td>Which Access Module Executes</td>
<td>127</td>
</tr>
<tr>
<td>Test Versions</td>
<td>127</td>
</tr>
<tr>
<td>Debugging the Application</td>
<td>128</td>
</tr>
<tr>
<td>Command Facility</td>
<td>128</td>
</tr>
<tr>
<td>SQL Trace Facility</td>
<td>129</td>
</tr>
<tr>
<td>EXPLAIN Statement</td>
<td>130</td>
</tr>
<tr>
<td>Online Debugger</td>
<td>130</td>
</tr>
</tbody>
</table>

### SQL Programming Techniques

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modularized Programming</td>
<td>130</td>
</tr>
<tr>
<td>Sharing a Cursor</td>
<td>131</td>
</tr>
<tr>
<td>Requirements</td>
<td>131</td>
</tr>
<tr>
<td>Verifying External Cursors</td>
<td>132</td>
</tr>
<tr>
<td>Shared Cursor Example</td>
<td>132</td>
</tr>
<tr>
<td>Using the SET ACCESS MODULE Statement</td>
<td>133</td>
</tr>
<tr>
<td>Why You Use It</td>
<td>133</td>
</tr>
<tr>
<td>Default Access Module Specification</td>
<td>133</td>
</tr>
<tr>
<td>When to Issue SET ACCESS MODULE</td>
<td>134</td>
</tr>
<tr>
<td>Using a Host Variable</td>
<td>134</td>
</tr>
<tr>
<td>Pseudoconversational Programming</td>
<td>135</td>
</tr>
<tr>
<td>Updating After a Pseudoconverse</td>
<td>135</td>
</tr>
<tr>
<td>Using SUSPEND SESSION and RESUME SESSION</td>
<td>135</td>
</tr>
<tr>
<td>What SUSPEND SESSION Does</td>
<td>135</td>
</tr>
<tr>
<td>What RESUME SESSION Does</td>
<td>136</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Executing Prepared CALL Statements</td>
<td>162</td>
</tr>
<tr>
<td>What to Do</td>
<td>162</td>
</tr>
<tr>
<td>Declaring a Cursor</td>
<td>162</td>
</tr>
<tr>
<td>Preparing the Statement</td>
<td>162</td>
</tr>
<tr>
<td>Building the Statement Text</td>
<td>162</td>
</tr>
<tr>
<td>Declaring a Host Variable Array</td>
<td>162</td>
</tr>
<tr>
<td>Executing the Fetch</td>
<td>163</td>
</tr>
<tr>
<td>Sample Program</td>
<td>159</td>
</tr>
<tr>
<td>Executing Prepared SELECT Statements</td>
<td>158</td>
</tr>
<tr>
<td>What to Do</td>
<td>158</td>
</tr>
<tr>
<td>Declaring a Cursor</td>
<td>158</td>
</tr>
<tr>
<td>Preparing the Statement</td>
<td>158</td>
</tr>
<tr>
<td>Building the Statement Text</td>
<td>158</td>
</tr>
<tr>
<td>Declaring a Host Variable Array</td>
<td>159</td>
</tr>
<tr>
<td>Executing the Fetch</td>
<td>159</td>
</tr>
<tr>
<td>Sample Program</td>
<td>159</td>
</tr>
<tr>
<td>Dynamic SQL</td>
<td>152</td>
</tr>
<tr>
<td>To Insert, Update, or Delete</td>
<td>152</td>
</tr>
<tr>
<td>To Select</td>
<td>153</td>
</tr>
<tr>
<td>To CALL an SQL Invoked Procedure</td>
<td>153</td>
</tr>
<tr>
<td>Dynamic Insert, Update, and Delete Operations</td>
<td>154</td>
</tr>
<tr>
<td>Using EXECUTE IMMEDIATE</td>
<td>154</td>
</tr>
<tr>
<td>When to Use It</td>
<td>154</td>
</tr>
<tr>
<td>EXECUTE IMMEDIATE example</td>
<td>154</td>
</tr>
<tr>
<td>Using PREPARE</td>
<td>155</td>
</tr>
<tr>
<td>Why You Use PREPARE</td>
<td>155</td>
</tr>
<tr>
<td>Determining Information About the Prepared Statement</td>
<td>155</td>
</tr>
<tr>
<td>Declaring SQLDA</td>
<td>156</td>
</tr>
<tr>
<td>Declaring SQLDA in CA ADS</td>
<td>156</td>
</tr>
<tr>
<td>PREPARE Example</td>
<td>157</td>
</tr>
<tr>
<td>Error-checking</td>
<td>157</td>
</tr>
<tr>
<td>Using EXECUTE</td>
<td>157</td>
</tr>
<tr>
<td>Why You Use EXECUTE</td>
<td>157</td>
</tr>
<tr>
<td>EXECUTE Example</td>
<td>157</td>
</tr>
<tr>
<td>Error-checking</td>
<td>157</td>
</tr>
<tr>
<td>Repeating EXECUTE</td>
<td>157</td>
</tr>
<tr>
<td>Executing Prepared SELECT Statements</td>
<td>158</td>
</tr>
<tr>
<td>What to Do</td>
<td>158</td>
</tr>
<tr>
<td>Declaring a Cursor</td>
<td>158</td>
</tr>
<tr>
<td>Preparing the Statement</td>
<td>158</td>
</tr>
<tr>
<td>Building the Statement Text</td>
<td>158</td>
</tr>
<tr>
<td>Declaring a Host Variable Array</td>
<td>159</td>
</tr>
<tr>
<td>Executing the Fetch</td>
<td>159</td>
</tr>
<tr>
<td>Sample Program</td>
<td>159</td>
</tr>
</tbody>
</table>
Sample Program .......................................................................................................................... 163
Dynamic SQL Caching .................................................................................................................. 166
Searching the Cache ...................................................................................................................... 166
Impact of Database Definition Changes ...................................................................................... 167
SQL-Defined Databases and Caching .......................................................................................... 167
Non-SQL-Defined Databases and Caching .................................................................................. 168
Controlling the Cache ................................................................................................................... 168

Sample JCLs ................................................................................................................................. 169
z/OS ............................................................................................................................................. 169
   Central Version JCL .................................................................................................................. 169
   Local Mode JCL ....................................................................................................................... 170
   Variable Definitions ................................................................................................................. 172
z/VSE .......................................................................................................................................... 173
   Central Version JCL .................................................................................................................. 173
   Variable Definitions ................................................................................................................. 174
   Local Mode JCL ....................................................................................................................... 174
   Usage ...................................................................................................................................... 175
      IDMSLBLs Procedure .......................................................................................................... 175
      Logical Unit Assignments ................................................................................................... 175
      COBOL Internal Sort .......................................................................................................... 175
z/VM ........................................................................................................................................... 175
   Commands for Central Version Execution .............................................................................. 176
   Variable Definitions ................................................................................................................. 176
   Usage ...................................................................................................................................... 177
      Local Mode .......................................................................................................................... 177
      SYSIPT File ......................................................................................................................... 178
      SYSIDMS File ...................................................................................................................... 178

Test Database ............................................................................................................................. 179
Table Names and Descriptions ..................................................................................................... 179
   ASSIGNMENT .................................................................................................................... 180
   BENEFITS ............................................................................................................................ 180
   CONSULTANT ..................................................................................................................... 181
   COVERAGE ............................................................................................................................ 181
   DEPARTMENT ...................................................................................................................... 181
   DIVISION .............................................................................................................................. 182
   EMPLOYEE ........................................................................................................................... 182
   EXPERTISE ........................................................................................................................... 182
   INSURANCE_PLAN ................................................................................................................. 183
Precompiler Directives ................................................................. 221

Overriding DDLDM Area Ready Mode ................................................. 221
  Syntax .................................................................................. 221
  Parameters ........................................................................ 221

No Logging of Program Activity Statistics ........................................ 222
  Syntax .................................................................................. 222
  Parameters ........................................................................ 222

Generating a Source Listing ............................................................. 222
  Syntax .................................................................................. 222
  Parameters ........................................................................ 222
  Usage .................................................................................. 223
Programming IDMS SQL

The topics in this section are for CA IDMS users responsible for designing and developing application programs using embedded SQL. It documents aspects of CA IDMS that are specific to application programming with SQL, including pre-compiler options and data type conversions between the database and the program language. These topics are for a user with application programming experience and a working knowledge of SQL. Users should also be familiar with concepts of CA IDMS.

How Examples are Presented

All program examples are in COBOL, unless otherwise indicated.

Most examples of access to an SQL-defined database refer to a test database (see page 179) of employee information that is supplied as part of CA IDMS installation. The term CA IDMS refers to any one of the following 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

The actual product names are used for CA IDMS/DB, CA IDMS/DC, CA IDMS UCF, DC/UCF, and CA IDMS DDS to identify the specific CA IDMS component only when it is important to your understanding of the product.

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. You can use SQL to access both SQL-defined and network-defined databases.

- SQL Database Access (see page 15)
- Network-defined Database Access (see page 19)

SQL Database Access

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

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.

**Local Variables**

A local variable of an SQL routine is a program variable declared in a compound statement of an SQL routine. Local variables can be used to receive data retrieved from the database and to supply data to be added to the database.

**Routine Parameter**
A routine parameter of an SQL routine is a program variable declared in the parameter definition of an SQL routine. Routine parameters provide for the mechanism of passing data between an SQL routine and its invoker, but they can also be used to receive data retrieved from the database and to supply data to be added to the database.

**CALL**

The CALL procedure is the SQL statement that invokes an external procedure's program or an SQL procedure using a remote procedure paradigm. Input values are passed from CA IDMS to the program or SQL procedure. The output values are returned into the host variables of the program or into the local variables or routine parameters of the SQL procedure specified in the procedure reference.

**Bulk Processing**

Bulk processing is a CA IDMS extension to the SQL standard 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 program can prepare, or compile, certain SQL statements at runtime. This allows the program to execute an SQL statement that is not known until runtime.

**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 following example 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

<table>
<thead>
<tr>
<th>DEPT_ID</th>
<th>DEPT_NAME</th>
<th>DIV_CODE</th>
<th>DEPT_HEAD_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3510</td>
<td>APPRAISAL - USED CARS</td>
<td>D02</td>
<td>3082</td>
</tr>
<tr>
<td>4500</td>
<td>HUMAN RESOURCES</td>
<td>D09</td>
<td>3222</td>
</tr>
<tr>
<td>2210</td>
<td>SALES - NEW CARS</td>
<td>D04</td>
<td>2010</td>
</tr>
<tr>
<td>5000</td>
<td>CORPORATE ACCOUNTING</td>
<td>D09</td>
<td>2466</td>
</tr>
<tr>
<td>3520</td>
<td>APPRAISAL NEW CARS</td>
<td>D04</td>
<td>3769</td>
</tr>
<tr>
<td>4600</td>
<td>MAINTENANCE</td>
<td>D06</td>
<td>2096</td>
</tr>
<tr>
<td>4200</td>
<td>LEASING - NEW CARS</td>
<td>D04</td>
<td>1003</td>
</tr>
<tr>
<td>5100</td>
<td>BILLING</td>
<td>D06</td>
<td>2598</td>
</tr>
<tr>
<td>6000</td>
<td>LEGAL</td>
<td>D09</td>
<td>1003</td>
</tr>
<tr>
<td>1100</td>
<td>PURCHASING - USED CARS</td>
<td>D02</td>
<td>2246</td>
</tr>
<tr>
<td>3530</td>
<td>APPRAISAL SERVICE</td>
<td>D06</td>
<td>2209</td>
</tr>
<tr>
<td>5200</td>
<td>CORPORATE MARKETING</td>
<td>D09</td>
<td>2804</td>
</tr>
<tr>
<td>1110</td>
<td>PURCHASING - NEW CARS</td>
<td>D04</td>
<td>1765</td>
</tr>
<tr>
<td>3000</td>
<td>CUSTOMER SERVICE</td>
<td>D09</td>
<td>4321</td>
</tr>
<tr>
<td>6200</td>
<td>CORPORATE ADMINISTRATION</td>
<td>D09</td>
<td>2461</td>
</tr>
<tr>
<td>2200</td>
<td>SALES - USED CARS</td>
<td>D02</td>
<td>2180</td>
</tr>
<tr>
<td>1120</td>
<td>PURCHASING - SERVICE</td>
<td>D06</td>
<td>2004</td>
</tr>
<tr>
<td>4900</td>
<td>MIS</td>
<td>D09</td>
<td>2466</td>
</tr>
</tbody>
</table>

### EMPLOYEE (DEPT_ID) references DEPARTMENT (DEPT_ID)

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMP_LNAME</th>
<th>EMP_ID</th>
<th>(DEPT_HEAD_ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>FORDMAN</td>
<td>5008</td>
<td>1100</td>
</tr>
<tr>
<td>1100</td>
<td>HALLORAN</td>
<td>4703</td>
<td>1100</td>
</tr>
<tr>
<td>1100</td>
<td>HAMEL</td>
<td>2246</td>
<td>1100</td>
</tr>
<tr>
<td>1110</td>
<td>ALEXANDER</td>
<td>1765</td>
<td>1110</td>
</tr>
<tr>
<td>1110</td>
<td>WIDMAN</td>
<td>2106</td>
<td></td>
</tr>
<tr>
<td>1120</td>
<td>JOHNSON</td>
<td>2004</td>
<td></td>
</tr>
<tr>
<td>1120</td>
<td>JOHNSON</td>
<td>3294</td>
<td></td>
</tr>
<tr>
<td>1120</td>
<td>UMIDY</td>
<td>2898</td>
<td></td>
</tr>
</tbody>
</table>
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

- **Not 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.

Network-defined Database Access

CA IDMS provides the ability to use SQL to access a network-defined database. The SQL statements used to access this 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.

Note: For more information on accessing network-defined databases using SQL, see Accessing Network-Defined Databases (https://docops.ca.com/display/IDMSCU/Accessing+Network-Defined+Databases).

SQL Application Development

Contents

- Writing the Application (see page 20)
- Creating Executable Modules (see page 20)
- Executing the Application (see page 23)
- Testing and Debugging the Application (see page 23)

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

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

Note: For more information on the complete syntax for all CA IDMS SQL statements, see Statements (https://docops.ca.com/display/IDMSCU/Statements).

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

For more information on how you take these steps, see Preparing and Executing the Program (see page 114).
The next flow chart shows the result of each step in the process:

1. **Program with embedded SQL**
   - Precompile

   - Program source file
     - Compile
       - Object file
         - Linkage edit
           - Program load module
             - Load library

   - Error listing

   - RCM
     - Access module
       - Dictionary

   - Optimize
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.

Concurrent 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.

Note: For more information on specifying isolation level and ready mode, see Concurrency Control and Isolation Levels. (see page 41)

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.

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.
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 on the executing program and about SQL statement execution
  
  **Note:** For more information on displaying SQLCA fields, see SQL Status Checking and Error Handling (see page 44).

- Requesting a trace of all SQL commands issued from a batch application
  
  **Note:** For more information on the SQL trace facility, see Debugging the Application (see page 128).

- Issuing GET DIAGNOSTICS SQL statements to request diagnostic information from the DBMS about the last executed SQL statement
  
  **Note:** For more information on the GET DIAGNOSTICS statement, see GET DIAGNOSTICS (https://docops.ca.com/display/IDMSCU/GET+DIAGNOSTICS).

Writing an SQL Program

Databases can be accessed with SQL using any of the following methods:

- Host Variables in an SQL Statement (see page 25)
- Local Variables and Routine Parameters (see page 29)
- SQL Sessions (see page 30)
- Database Transactions - IDMS SQL (see page 33)
- Effect of Teleprocessing Statements and Events (see page 39)
- Concurrency Control and Isolation Levels (see page 41)
- SQL Status Checking and Error Handling (see page 44)
Host Variables in an SQL Statement

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 embedded in application programs.

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
- Supply information for dynamically executed statements. For more information, see Section 8, Using Dynamic SQL.

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:

```sql
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.

The following topics are discussed on this page:

- SQL Declare Sections (see page 26)
- INCLUDE TABLE Directive (see page 27)
- Referring to Host Variables (see page 29)

SQL Declare Sections

In SQL Standard, 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.
```

A CA IDMS extension of the SQL standard 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.
```
INCLUDE TABLE Directive

INCLUDE TABLE Statement

You can use the INCLUDE TABLE statement, a CA IDMS extension of the SQL standard, 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, view, table procedure, procedure or function, 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 following INCLUDE statement directs the precompiler to define host variables for the DIVISION table, which has columns DIV_CODE, DIV_NAME, and DIV_HEAD_ID:

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

Structure Example

When the precompiler processes the INCLUDE TABLE statement in the prior example, it defines this structure:

```sql
*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).
    * SQLIND.
    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 perform the following:

- 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 or suffix for an element name
Direct the precompiler to generate a multiply-occurring array

**Note:** For more information on INCLUDE statement syntax and options, see INCLUDE (https://docops.ca.com/display/IDMSCU/INCLUDE).

### 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.

**Note:** For more information on bulk processing, see Bulk Processing (see page 67).

### Host Variable Array Structure

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

```
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).
      *                      SQLIND.
    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 the SQL standard
- The host variable declaration is for temporary table columns
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 following example, 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.

```sql
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.
```

Local Variables and Routine Parameters

Local variables and routine parameters are program variables of SQL routines. These variables can be used as any program variable and are necessary for the SQL routine to receive data from the database and to modify data in the database. In addition to their role as program variables, routine parameters are mainly used to pass input values from and output values to the invoker of the SQL routine.

Local variables are defined in the DECLARE statement of a compound SQL statement. Routine parameters are defined in the parameter-definition clause of the CREATE PROCEDURE or CREATE FUNCTION statements.

How Local Variables and Routine Parameters Are Used

Local variables and routine parameters are used as follows:
- 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
- Supply information for dynamically executed statements

Local Variable Example

In the following example, DEPT_ID, EMP_LNAME, and EMP_ID are local variables defined in a compound statement with label MAIN_BLOCK. DEPT_ID and EMP_LNAME receive column values and EMP_ID supplies a column value used in the search condition of the statement:

```sql
SELECT EMPLOYEE.DEPT_ID, EMPLOYEE.EMP_LNAME INTO MAIN_BLOCK.DEPT_ID,
     MAIN_BLOCK.EMP_LNAME
FROM EMPLOYEE
WHERE EMPLOYEE.EMP_ID = MAIN_BLOCK.EMP_ID;
```

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. Local variables and routine parameters are always nullable. However, as these are SQL variables, null support is built-in and null indicators must not be used.

⚠️ Note: For more information, see Null Values (https://docops.ca.com/display/IDMSCU/Null+Values).

SQL Sessions

- Beginning and Ending an SQL Session (see page 30)

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.

Beginning and Ending an SQL Session

Beginning an SQL Session

An SQL session begins when the program submits its first SQL statement. If that statement is a CONNECT, the session is connected to the dictionary specified by the statement and the session is said to be explicitly connected.

If the first statement is not a CONNECT, the session is automatically connected to a default dictionary.
Session Hierarchy

A hierarchy of database sessions occurs when an SQL invoked routine (an SQL procedure, table procedure, or function) starts its own session to access the database.

A database session that is started by a program executing as part of an SQL invoked routine is a subordinate session since it is under the control of the SQL session within which the routine was invoked. The controlling session is referred to as the subordinate session's encompassing session. A top-level session is one that has no encompassing session.

<table>
<thead>
<tr>
<th>Top-level/Encompassing Session</th>
<th>Application issues:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encompassing Session</td>
<td>-- CONNECT...</td>
</tr>
<tr>
<td>Subordinate Session</td>
<td>-- CALL &quot;PROCA&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encompassing and Subordinate Session</th>
<th>PROCA issues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-- CONNECT...</td>
</tr>
<tr>
<td></td>
<td>-- Invokes function FUNCB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subordinate Session</th>
<th>FUNCB issues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-- BIND RUNUNIT...</td>
</tr>
</tbody>
</table>

Default Dictionary

When establishing an automatically connected SQL session, CA IDMS connects the session to a default dictionary.

The default dictionary for a top-level session is established by:

- SYSIDMS DICTNAME parameter for a batch application
- Value of the DICTNAME attribute for the user session, as set by one of the following:
  - User profile
  - System profile
  - Default dictionary defined by the DBNAME table
  - DCUF SET DICTNAME system task
  - Call to IDMSIN01 to set the DICTNAME attribute
The default dictionary for a subordinate session is determined by the initiating routine definition's DEFAULT DATABASE parameter.

- If DEFAULT DATABASE CURRENT was specified, the default dictionary is the dictionary to which the encompassing SQL session is connected.

- If DEFAULT DATABASE NULL was specified (or defaulted), the default dictionary is determined in the same way as for a top-level session.

**Note:** For more information on the DEFAULT DATABASE parameter of the CREATE PROCEDURE, CREATE TABLE PROCEDURE or CREATE FUNCTION statements, see Statements (https://docops.ca.com/display/IDMSCU/Statements).

**SQL Statements that End a Session**

If the SQL session began automatically (that is, no CONNECT 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 to start the session, it ends when the program issues one of these statements:

- COMMIT RELEASE
- ROLLBACK RELEASE
- RELEASE

If an encompassing session ends, all of its subordinate sessions end also.

**Automatic Session Termination**

If a batch application program terminates execution by returning control to the operating system, SQL sessions still in progress are terminated automatically as if the application had issued a ROLLBACK RELEASE statement.
If a program returns control to a teleprocessing system or issues certain teleprocessing statements, such as FINISH TASK, SQL sessions still in progress may or may not be terminated depending on the event or statement issued and whether the session is suspended.

Note: For more information on the effect of teleprocessing statements on SQL sessions, see Effect of Teleprocessing Statements and Events (see page 39).

Database Transactions - IDMS SQL

A database transaction is a unit of recovery representing work done by one or more database sessions. All access to CA IDMS data from within an SQL session is done under the control of a database transaction. Transactions can be associated with one or more database sessions. A transaction can be associated with more than one session only if a session is eligible to share its transaction with other sessions. Transactions started by sessions that are not eligible to share their transaction are called nonshareable transactions.

- Managing Nonshareable Transactions (see page 33)
- Committing Changes (see page 34)
- Sharing Transactions Among Sessions (see page 34)
- Application Programming Considerations (see page 37)

Managing Nonshareable Transactions

Beginning a Transaction

A nonshareable transaction is started when the program submits an SQL statement that results in access to either user data or a dictionary, unless the session is already associated with a transaction.

Transaction Hierarchy

Just as sessions can be related in a hierarchical way, their associated transactions can also be related hierarchically. If a session is subordinate to another session, its transaction is subordinate to the encompassing session’s transaction. For more information on session hierarchies, see Beginning and Ending an SQL Session (see page 30).

When a transaction is committed or rolled back, all of its direct and indirect subordinates are also committed or rolled back.

Ending a Transaction

If a session’s transaction is not shareable, it ends when:

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

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

### Committing Changes

Changes made through an SQL session are committed when an SQL COMMIT statement is executed or when a teleprocessing statement is executed that results in the committing of database updates. If changes are not committed in one of these ways, updates made through an SQL session are backed out, either as the result of an explicit ROLLBACK request or automatically as the result of a teleprocessing statement or event.

**Note:** For more information on the effect of teleprocessing statements on database transactions, see Effect of Teleprocessing Statements and Events (see page 39).

Transaction sharing impacts the committing of database changes.

**Note:** For more information on the impact that sharing database transactions has on committing changes, see Sharing Transactions Among Sessions.

### Preserving Session State after a Commit

Normally when a transaction is committed, the state of the session is reset: cursors are closed, prepared statements are deleted and temporary tables are dropped. However, a CA IDMS extension to the SQL standard allows you to commit updates but preserve the session state as it was prior to the commit. This extension is the CONTINUE parameter of the COMMIT statement:

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

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

### Sharing Transactions Among Sessions

**Sharing a Transaction**

A transaction can be shared by multiple database sessions, both SQL sessions and non-SQL sessions (rununits). By sharing a transaction, sessions will not deadlock among themselves even if they access and update the same data.
Enabling Transaction Sharing

An SQL session is eligible to share its transaction if transaction sharing is in effect when the database session is started.

Transaction sharing is in effect for a top-level session if:

- TRANSACTION_SHARING=ON is specified in the SYSIDMS file for a batch application.

  Note: For more information on SYSIDMS parameters, see SYSIDMS Parameter Descriptions (https://docops.ca.com/display/IDMS19/SYSIDMS+Parameter+Descriptions).

- IDMSCINT or CICSOPT parameter specified TXNSHR=ON for CICS applications.

  Note: For more information on IDMSCINT and CICSOPT parameters, see Administrating CA IDMS System Operations (https://docops.ca.com/display/IDMS19/Administrating+CA+IDMS+System+Operations).

- Transaction sharing has been enabled for the executing DC/UCF task by means of a SYSGEN or DCMT command.

- Transaction sharing has been enabled though a call to IDMSIN01.

For subordinate sessions, transaction sharing is controlled through the TRANSACTION SHARING parameter of the SQL invoked routine’s definition unless overridden by a call to IDMSIN01 from within the routine.

- If TRANSACTION SHARING ON is specified, transaction sharing is enabled for all sessions started by the routine.

- If TRANSACTION SHARING OFF is specified, transaction sharing is disabled for all sessions started by the routine.

- If TRANSACTION SHARING DEFAULT is specified (or defaulted), the transaction sharing state that was in effect before the routine was called applies to all sessions started by the routine.

  Note: For more information on the TRANSACTION SHARING parameter of the CREATE PROCEDURE, CREATE TABLE PROCEDURE or CREATE FUNCTION statements, see Statements (https://docops.ca.com/display/IDMSCU/Statements).

Whether transaction sharing is enabled for a remote SQL session is determined by the attribute in effect in the CA IDMS environment in which the session-initiating statement is issued. (A remote session is one that is connected to a dictionary residing on a central version different from where the application is executing.)
Regardless of how transaction sharing is enabled, if it is in effect at the time a database session is started, then that database session is eligible to share its transaction with other database sessions started by the same task or user session. The following rules determine whether two sessions will share a transaction.

- A top-level database session will share its transaction with another top-level session if they are both eligible for transaction sharing.

- A subordinate database session that is eligible for transaction sharing will share its encompassing session's transaction even if the encompassing session is not eligible to share its transaction.
Application Programming Considerations

Transaction sharing affects applications in the following ways:

- An update made through a database session may impact other database sessions sharing the same transaction.

- A rollback issued within one database session affects all sessions that share the same transaction.

- A commit issued by a database session whose transaction is shared has no affect on the transaction unless all other sharing sessions have also been committed.

Inter-session Conflicts
Database sessions that share a transaction can impact each other in ways that would not be possible without transaction sharing since locking would prevent such interactions. For example, a record can be deleted by one database session while it is current of another database session that is sharing the same transaction. This can result in new and possibly unexpected error conditions. If a database session’s currency is impacted by an update made through another database session, that currency is invalidated. If a subsequent DML request, such as a FETCH from a cursor, relies on that invalidated currency, an error is returned.

- For SQL, the application receives an SQLCODE of -4 (statement failure) and an SQLRSN of 1087 (conflicting activity within a shared transaction).
- For navigational DML, an error status of xx03 is returned to the application.

Before enabling transaction sharing for an application, you should ensure that affected programs handle these errors appropriately.

**Effect of Rollback Requests**

If multiple database sessions share a transaction and one of those sessions issues a rollback request, all changes made within the transaction are immediately rolled out. Other sessions sharing the transaction must issue their own rollback requests before issuing any other DML requests. Issuing another DML request instead of a rollback will result in an error:

- For SQL, the application receives an SQLCODE of -5 (transaction failure) and an SQLRSN of 1088 (transaction forced to backout)
- For navigational DML, the run unit is terminated and an error status of xx19 is returned to the application.

**Effect of Commit Requests**

If multiple top level database sessions share a transaction and one of those sessions issues a commit request, no changes are committed until:

- All top-level sharing sessions that have had activity since the last commit, rollback or start of a transaction have issued a commit, or,
- Until a teleprocessing commit is issued.

The term "commit" refers to any DML command that would normally result in committing changes (COMMIT RELEASE, COMMIT CONTINUE, FINISH, and so forth).

A commit issued through a subordinate session has no impact on its transaction if it is shared since such a transaction can only be committed through the encompassing session.

Unless a COMMIT CONTINUE request is issued (for which currency locks are retained), all currencies owned by the issuing database session are immediately released. However, implicit exclusive locks and explicit locks acquired by the database session remain until the transaction is committed, even if the request terminates the database session.
Effect of Teleprocessing Statements and Events

Effect of Task-level DML Statements and Events

In a batch or DC/UCF environment, task-level commit and rollback statements and task-termination events affect the status of database transactions and SQL sessions, as the following table shows. Their effect on a subordinate SQL session is the same as their effect on its encompassing session.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Effect on Top-level SQL Sessions</th>
<th>Effect on Top-level Database Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMIT TASK</td>
<td>Equivalent to issuing a COMMIT CONTINUE on all nonsuspended SQL sessions</td>
<td>Commits changes made through all transactions except nonshareable transactions whose session is suspended.</td>
</tr>
<tr>
<td>COMMIT TASK ALL</td>
<td>Equivalent to issuing a COMMIT on all nonsuspended SQL sessions.</td>
<td>Commits changes made through all transactions except nonshareable transactions whose session is suspended.</td>
</tr>
<tr>
<td>FINISH TASK</td>
<td>Equivalent to issuing a COMMIT RELEASE on all nonsuspended SQL sessions</td>
<td>Commits changes made through all transactions except nonshareable transactions whose session is suspended.</td>
</tr>
<tr>
<td>ROLLBACK TASK CONTINUE</td>
<td>Equivalent to issuing a ROLLBACK on all nonsuspended SQL sessions. All suspended sessions whose shareable transaction is rolled back are marked as requiring rollback.</td>
<td>Rolls back changes made through all transactions except nonshareable transactions whose session is suspended.</td>
</tr>
<tr>
<td>ROLLBACK TASK</td>
<td>Equivalent to issuing a ROLLBACK RELEASE on all nonsuspended SQL sessions. All suspended sessions whose shareable transaction is rolled back are marked as requiring rollback.</td>
<td>Rolls back changes made through all transactions except nonshareable transactions whose session is suspended.</td>
</tr>
<tr>
<td>Normal task termination</td>
<td>Equivalent to issuing a ROLLBACK RELEASE on all nonsuspended SQL sessions. All suspended sessions whose shareable transaction is rolled back are marked as requiring rollback.</td>
<td>Rolls back changes made through all transactions except those for which all associated sessions are suspended.</td>
</tr>
<tr>
<td>Abnormal Task Termination</td>
<td>Equivalent to issuing a ROLLBACK RELEASE on all SQL sessions</td>
<td>Rolls back updates made by all transactions associated with the task or user session.</td>
</tr>
</tbody>
</table>

A task-level commit or rollback statement has no affect on transactions whose database sessions are suspended and for which transaction sharing is not in effect.
CICS Syncpoint and Backout Operations

The effect of a CICS syncpoint or backout operation on an SQL session depends on the parameters used to generate the version of the IDMSCINT interface module with which the program was link-edited and the CICSOPT parameters used to generate its corresponding IDMSINTC interface module.

The options in effect for a program that starts an SQL session determine how that session and its transaction are impacted by CICS syncpoint and backout operations. The parameters that impact their semantics are:

- **AUTOCMT**: Enabling this option makes the work done by the database session eligible to be included in a CICS UOW (Unit of Work). If included, CICS syncpoint and backout operations affect the changes made by the session. Whether the changes made by a session are actually included in the CICS UOW is determined by the AUTONLY setting and whether the application issues its own commit or rollback DML requests prior to the CICS syncpoint or backout operation.

- **AUTONLY**: Enabling this option in conjunction with the AUTOCMT option forces the work done by the database session to be included in the CICS UOW. DML statements that would normally commit work (such as FINISH TASK or COMMIT) do not cause changes to be committed even if the session itself is terminated. The session's changes are committed only when the CICS syncpoint occurs. On the other hand, if the changes made by a session for which AUTONLY is enabled are backed out, either as the result of a DML ROLLBACK request or because of some environmental condition such as a deadlock, the entire CICS UOW will eventually be backed out. This ensures consistent behavior across all resources updated by the application.

If AUTONLY is not enabled but AUTOCMT is, the work done by the database session is included in the CICS UOW only if the application does not issue commit or rollback DML requests prior to the CICS syncpoint operation.

Enabling AUTONLY without AUTOCMT has no impact on syncpoint operations.

**Note:** If transaction sharing is enabled, AUTONLY and AUTOCMT are always enabled.

- **ONCOMT**: This option specifies the effect that a CICS syncpoint operation has on a database session whose work is included in the CICS UOW. The session can optionally be treated as if a COMMIT RELEASE, COMMIT, or COMMIT CONTINUE were issued, meaning that it can be terminated, remain active but have currencies cleared or remain active with currencies left intact.

- **ONBACK**: This option specifies the effect that a CICS backout operation has on a database session whose work is included in the CICS UOW. The session can optionally be treated as if a ROLLBACK RELEASE or a ROLLBACK were issued, meaning that it can be terminated or remain active but have its currencies cleared.

All of these options can be specified through both IDMSCINT and CICSOPT parameters. The CICSOPT parameters can either override their IDMSCINT counterparts or be used as a default.

**Note:** For more information on these parameters, see Administrating CA IDMS System Operations (https://docops.ca.com/display/IDMS19/Administrating+CA+IDMS+System+Operations).
A CICS syncpoint operation occurs when a CICS SYNCPOINT statement is executed by the application and when the CICS task terminates normally. A CICS backout operation occurs when a CICS BACKOUT statement is executed by the application and when the CICS task terminates abnormally.

The following table summarizes the impact of CICS syncpoint and backout operations and task-termination events on SQL sessions and their transactions.

<table>
<thead>
<tr>
<th>Operation or Event</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNCPOINT Operation</td>
<td>If AUTOCMT is not in effect for a session, the SYNCPOINT operation has no impact on the session and its transaction. If AUTOCMT is in effect for a session, the uncommitted changes made by the session are committed. The impact on the session is determined by the session's ONCOMT option.</td>
</tr>
<tr>
<td>BACKOUT Operation</td>
<td>If AUTOCMT is not in effect for a session, the BACKOUT operation has no impact on the session and its transaction. If AUTOCMT is in effect for a session, the uncommitted changes made by the session are backed out. The impact on the session is determined by the session's ONBACK option.</td>
</tr>
<tr>
<td>Normal CICS Task Termination</td>
<td>All nonsuspended SQL sessions are treated as if a ROLLBACK RELEASE were issued (although their changes may have been committed by the preceding syncpoint operation). Their uncommitted changes are backed out.</td>
</tr>
<tr>
<td>Abnormal CICS Task Termination</td>
<td>All SQL sessions are treated as if a ROLLBACK RELEASE were issued. Their uncommitted changes are backed out.</td>
</tr>
</tbody>
</table>

**Effect of Task-level DML Statements in CICS**

In a CICS environment, task-level commit and rollback statements have the same effect on sessions as in a DC/UCF environment. However, a task-level commit request (COMMIT TASK, COMMIT TASK ALL, or FINISH task) does not commit the work done by sessions whose AUTONLY and AUTOCMT options are enabled.

Just as in a DC/UCF environment, a task-level rollback request (ROLLBACK TASK or ROLLBACK TASK CONTINUE) affects all transactions except nonshareable transactions whose session is suspended.

**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 following descriptions explain 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 does not prevent other database sessions that are sharing the same transaction from updating a session's current cursor position or its most-recently retrieved row of a single row select.

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.

⚠️ **Note:** For more information on how to create the access module, see Creating the Access Module (see page 120).

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.

⚠️ **Note:** For more information on the SET TRANSACTION statement, see SET TRANSACTION (https://docops.ca.com/display/IDMSCU/SET+TRANSACTION).

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.

⚠️ **Note:** For more information on the READY parameter of the CREATE ACCESS MODULE statement, see CREATE ACCESS MODULE (https://docops.ca.com/display/IDMSCU/CREATE+ACCESS+MODULE).
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.

Note: For more information on the lock management system, see Administrating CA IDMS Database (https://docops.ca.com/display/IDMS19/Administrating+CA+IDMS+Database).

SQL Status Checking and Error Handling

When CA IDMS executes an SQL statement, it returns information on 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. An alternative to checking the SQLCA is the use of the GET DIAGNOSTICS statement that provides for enhanced diagnostic information.

This article includes the following topics:

- SQLCA (see page 44)
- Displaying SQL Communication Area Fields (see page 49)
- Error Handling (see page 49)
- Checking Specific Errors (see page 50)
- Using GET DIAGNOSTICS (see page 51)

SQLCA

The SQL Communication Area (SQLCA) is a data structure to which the DBMS returns information on 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 following list displays the SQLSTATE values that CA IDMS can return. It 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.

- **SQL standard values** -- Class and subclass values beginning with the characters A-H and 0-4 are established by the SQL 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.)

SQLSTATE Values

00 Successful completion
   000 No subclass

01 Warning
   000 No subclass
   004 String data, right truncation
   00C SQL-invoked procedure returned result sets
   00D Additional result sets returned
   00E Attempt to return too many result sets
   010 Column cannot be mapped
   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 both of DDLCAT/DDLDML areas
   610 Database is inconsistent with request
   611 SQL routine parse error
   612 ADS compilation for an SQL routine failed
   613 Drop of SQL routine completed with warnings
   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

0M Invalid SQL-invoked procedure reference
   000 No subclass

0N SQL/XML Mapping Error
   000 No subclass
   001 Unmappable XML name
   002 Invalid XML character

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
   009 Nonidentical notations with the same name
   00K Nonidentical unparsed entities with the same name
   00L Not an XML document
   00M Invalid XML document
   00N Invalid XML content
   00R XML value overflow
   00S Invalid comment
00T Invalid processing instruction
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

34 Invalid cursor name
000 No subclass

37 Syntax error or access rule violation
000 No subclass

38 External routine exception
000 No subclass
999 ADS dialog failed or dialog does not exist

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>The SQL statement returned an error (see the following error values)</td>
</tr>
<tr>
<td>0</td>
<td>The SQL statement was executed successfully</td>
</tr>
<tr>
<td>1</td>
<td>The SQL statement was executed successfully with a warning</td>
</tr>
<tr>
<td>100</td>
<td>There are no more rows associated with the current query, or no rows satisfied the search criteria in a searched update or delete</td>
</tr>
</tbody>
</table>

Note: The SQL standard only defines 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 following table 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Level of failure</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>Task</td>
<td>An internal error caused a task abend, leading to rollback and termination of the SQL transaction and termination of the SQL session</td>
</tr>
<tr>
<td>-6</td>
<td>SQL session</td>
<td>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: ROLLBACK, RELEASE, RELEASE. The equivalent TP monitor command is.</td>
</tr>
<tr>
<td>-5</td>
<td>SQL transaction</td>
<td>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: ROLLBACK.</td>
</tr>
</tbody>
</table>
Value of failure | Meaning
--- | ---
ROLLBACK RELEASE
RELEASE | The equivalent TP monitor command

<table>
<thead>
<tr>
<th>Field</th>
<th>Description of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCLNO</td>
<td>Source file line number from which the SQL statement was obtained</td>
</tr>
<tr>
<td>SQLCSER</td>
<td>Offset into the SQL statement buffer where a syntax error was recognized</td>
</tr>
<tr>
<td>SQLCNRP</td>
<td>Number of rows processed by the SQL statement</td>
</tr>
<tr>
<td>SQLCERM</td>
<td>Text of the error message</td>
</tr>
<tr>
<td>SQLCERL</td>
<td>Length of error message text</td>
</tr>
<tr>
<td>SQLCNRRS</td>
<td>Actual number of results sets that an SQL invoked procedure returns</td>
</tr>
</tbody>
</table>

**Note:** For more information on the documentation of DB messages, see [Messages Search](https://docops.ca.com/display/IDMS19/Messages+Search), or issue a DCMT DISPLAY MESSAGE DBnmmnnn statement, as documented in [IDMS System Tasks and Operator Commands](https://docops.ca.com/display/IDMSCU/IDMS+System+Tasks+and+Operator+Commands).

Other SQLCA Fields

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

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):

<table>
<thead>
<tr>
<th>Field</th>
<th>Description of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLDTS</td>
<td>Date and time the program was compiled</td>
</tr>
<tr>
<td>SQLPGM</td>
<td>Name of the RCM that is the source of the SQL statement</td>
</tr>
</tbody>
</table>

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).

⚠️ **Note:** For more information on the SQLCA structure, see Requirements and Options for Host Languages (see page 76).

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.

⚠️ **Note:** For more information on the requirements for calling IDMSIN01 to display SQL messages, see Callable Services Reference (https://docops.ca.com/display/IDMSCU/Callable+Services+Reference).

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:

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

### Overriding WHENEVER

Once 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

⚠️ **Note:** For more information on the WHENEVER statement, see [WHENEVER](https://docops.ca.com/display/IDMSCU/WHENEVER).

### 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 section discusses when to code specific error-checking logic in Section4, Data Manipulation with SQL, and other sections 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:

```sql
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.


Using GET DIAGNOSTICS

The use of GET DIAGNOSTICS instead or in addition to checking SQLCA offers the following advantages:

- Better portability because of the SQL standards compliance
- Availability of more diagnostic information
- Independent of host language
- Built-in formatting of all diagnostic information

Note: For more information on GET DIAGNOSTICS, see GET DIAGNOSTICS (https://docops.ca.com/display/IDMSCU/GET+DIAGNOSTICS).

Data Manipulation with SQL

When SQL is used in a host language program, you will need to perform data manipulation. There are several ways that the program can take advantage of SQL DML in CA IDMS.
SQL DML Statements

Use the following SQL statements in data manipulation operations:

- **SELECT** -- To retrieve data
- **INSERT** -- To add data
- **UPDATE** -- To modify data
- **DELETE** -- To delete data
- **CALL** -- To invoke an SQL invoked procedure or table procedure.

SQL data manipulation statements provide the following 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

Consequently, you have several options for performing each type of data manipulation.

- Retrieving Data (see page 52)
- Adding Data (see page 54)
- Modifying Data (see page 56)
- Deleting Data (see page 58)
- Using Indicator Variables in Data Manipulation (see page 59)

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**, local variables, or routine parameters 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:

  ```sql
  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:

  ```sql
  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 1, 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.

⚠️ **Note:** For more information on updating rows, see Modifying Data (see page 56).

### 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.

⚠️ **Note:** For more information on retrieving multiple rows, see Data Manipulation with SQL (see page 51).

### Adding Data

#### Using INSERT

In a program, you use an INSERT statement to add data to the database in one of the following 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:

```sql
EXEC SQL
INSERT INTO DIVISION
```
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:
Inserting Multiple Rows

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

Note: Referential constraints defined as linked clustered are not permitted to not cross page group boundaries.

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

Note: For information on positioned updates, see Using a Cursor (see page 61).
Checking UPDATE Status

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

⚠️ **Note:** For more information on checking statement execution for constraint violation, see Adding Data (see page 54).

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:

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

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

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

```sql
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

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

⚠️ Note: For more information on positioned deletes, see Using a Cursor (see page 61).

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

Important! 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.

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 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 next example, the program assigns 0 to the indicator variable after changing the value of the host variable VAC-TAKEN. CA IDMS stores 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
  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.

Using a Cursor

- Declaring a Cursor (see page 61)
- Fetching a Row (see page 62)
- Executing a Positioned Update or Delete (see page 65)

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.

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

⚠️ **Note:** For more information on all criteria that an updateable cursor must meet, see DECLARE CURSOR (https://docops.ca.com/display/IDMSCU/DECLARE+CURSOR).

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:

```sql
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.
```

Fetching a Row

Opening the Cursor

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

```sql
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:

```sql
EXEC SQL
FETCH EMP_SUM INTO :EMP-ID,
:MANAGER-ID :MANAGER-ID-I,
:EMP-FNAME,
:EMP-LNAME,
```
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 following example.

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:

```sql
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:

```
Declare
cursor

Open
cursor

Iterative
logic

Advance
cursor

Close
```
Executing a Positioned Update or Delete

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.

Note: For more information on pseudoconversational programming with embedded SQL, see Pseudoconversational Programming (see page 135).

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.
Positioned Update Example

In the following 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.

```sql
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*.

**Note:** For more information on checking the status of DELETE statements in general, see Data Manipulation with SQL (see page 51).

Positioned Delete Example

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

```sql
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.
```

Bulk Processing
Executing a Bulk Fetch (see page 68)
Executing a Bulk Select (see page 71)
Executing a Bulk Insert (see page 72)

A CA IDMS extension of the SQL standard 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.

⚠️ Note: For more information on declaring a host variable array in CA ADS, COBOL and PL/I see Requirements and Options for Host Languages (see page 76).

Executing a 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

⚠️ Note: For more information on the FETCH statement, see FETCH (https://docops.ca.com/display/IDMSCU/FETCH).

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 retrieves 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 retrieves 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 Using a Cursor (see page 61). The DBMS always sets the value of SQLCNRP equal to the number of rows returned unless an error occurs during processing.

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

<table>
<thead>
<tr>
<th>Result of bulk fetch</th>
<th>SQLCODE value</th>
<th>SQLCNRP value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No rows are returned</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>At least one row is returned but fewer rows than the maximum allowed</td>
<td>100</td>
<td>Equals the number of rows returned</td>
</tr>
<tr>
<td>The number of rows returned matches the maximum allowed</td>
<td>0</td>
<td>Equals the number of rows returned</td>
</tr>
</tbody>
</table>

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.

```sql
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.

Executing a 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 following table shows the possible combinations of SQLCODE and SQLCNRP values on a bulk select:

<table>
<thead>
<tr>
<th>Result of bulk select</th>
<th>SQLCODE value</th>
<th>SQLCNRP value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No rows are returned</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>At least one row is returned but fewer rows than</td>
<td>100</td>
<td>Greater than 0 and less than or equal to the maximum allowed</td>
</tr>
<tr>
<td>the maximum allowed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The number of rows returned exceeds the maximum</td>
<td>Less than 0</td>
<td>Equal to the maximum allowed</td>
</tr>
<tr>
<td>allowed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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.

Executing a 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:

```sql
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 following table shows the possible combinations of SQLCODE and SQLCNRP values on a bulk insert:

<table>
<thead>
<tr>
<th>Result of bulk insert</th>
<th>SQLCODE value</th>
<th>SQLCNRP value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fewer rows than the number of rows specified are inserted because the insert failed on a row</td>
<td>Less than 0</td>
<td>Equal to the relative row number of the failing row</td>
</tr>
<tr>
<td>The number of rows inserted matches the number of rows specified</td>
<td>0</td>
<td>Equal to the number of rows inserted</td>
</tr>
</tbody>
</table>

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.

Invoking Procedures
There are two types of SQL invoked procedures: a procedure and a table procedures. Both types can be invoked using either a CALL statement or a SELECT statement. This section describes the results of invoking procedures in each of these ways.

CALL Statement

In a program, you can use the CALL statement to invoke a (table) procedure. The following sections describe the results of invoking each type of procedure using a CALL statement.

---

**Note:** For more information on SQL procedures and table procedures, see Defining and Using Table Procedures (https://docops.ca.com/display/IDMSCU/Defining+and+Using+Table+Procedures).

---

**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.

**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 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.

Note: For more information on using cursors, see Using a Cursor (see page 61).

SELECT Statement

The SELECT statement can be used as an alternative to the CALL statement to invoke a (table) procedure. The following sections describe the results of invoking each type of procedure using a SELECT statement.

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.

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.

**Note:** For more information on using cursors, see Using a Cursor (see page 61).

Requirements and Options for Host Languages

There are requirements and options that apply to a particular host language when you embed SQL in an application program to access CA IDMS.

**Note:** The SQL Web Connect feature allows all IDMS customers limited use of dynamic SQL. The use of static, precompiled SQL requires a full SQL license.

This section presents information that is specific to embedding SQL in a CA ADS application program. Refer to the following manuals for documentation of all aspects of CA ADS application programming:

- Using SQL in a CA ADS Application (see page 82)
- Using SQL in a COBOL Application Program (see page 84)
- Using SQL in a PL/I Application Program (see page 102)

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**
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 following example 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**

```
  ++-+-+----+----+----+----+----+----+----+----+-
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**

- Do not insert a comment in the middle of a string constant or delimited identifier
- Do not use the CA ADS comment character `!` to insert a comment in an embedded SQL statement

**SQL Comment Example**

```
EXEC SQL
  INSERT INTO SKILL VALUES (5678, 'TELEMARKETING', 'PRESENT SALES SCRIP
T OVER THE TELEPHONE, INPUT RESULTS')
--!
END-EXEC.
```
The following example 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.
```

### 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.

**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.

⚠️ **Note**: For more information on ADSC and the MOVE command, see [ADS Reference](https://docops.ca.com/display/IDMSCU/ADS+Reference).

**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:

<table>
<thead>
<tr>
<th>CA ADS PICTURE and USAGE clause</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC X(n) USAGE DISPLAY</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>01 name 49 name-LEN PIC S9(4) COMP 49 name-TEXT PIC X(n)</td>
<td>VARCHAR(n)</td>
</tr>
<tr>
<td>PIC S9(p-s)V9(s) USAGE COMP-3</td>
<td>DECIMAL(p,s)</td>
</tr>
<tr>
<td>PIC 9(p-s)V9(s) USAGE COMP-3</td>
<td>UNSIGNED DECIMAL(p,s)¹</td>
</tr>
<tr>
<td>USAGE COMP-2</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>USAGE COMP-1</td>
<td>REAL</td>
</tr>
<tr>
<td>USAGE COMP-1</td>
<td>FLOAT</td>
</tr>
<tr>
<td>PIC S9(n) USAGE COMP (where n&lt;5)</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>PIC S9(n) USAGE COMP (where n&gt;4 and n&lt;10)</td>
<td>INTEGER</td>
</tr>
<tr>
<td></td>
<td>LONGINT or BIGINT</td>
</tr>
</tbody>
</table>
### CA ADS PICTURE and USAGE clause

<table>
<thead>
<tr>
<th>CA ADS PICTURE and USAGE clause</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(n) USAGE COMP (where n&gt;9)</td>
<td></td>
</tr>
<tr>
<td>PIC S9(p-s)V9(s) USAGE DISPLAY</td>
<td>NUMERIC(p,s)</td>
</tr>
<tr>
<td>PIC 9(p-s)V9(s) USAGE DISPLAY</td>
<td>UNSIGNED NUMERIC(p,s)¹</td>
</tr>
<tr>
<td>PIC X(n) USAGE DISPLAY</td>
<td>BINARY(n)</td>
</tr>
<tr>
<td>PIC G(n) USAGE DISPLAY-1</td>
<td>GRAPHIC(n)¹</td>
</tr>
</tbody>
</table>

01 name
49 name-LEN PIC S9(4) COMP
49 name-TEXT PIC G(n) DISPLAY-1

<table>
<thead>
<tr>
<th>CA ADS PICTURE and USAGE clause</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC X(10) USAGE DISPLAY</td>
<td>DATE</td>
</tr>
<tr>
<td>PIC X(8) USAGE DISPLAY</td>
<td>TIME</td>
</tr>
<tr>
<td>PIC X(26) USAGE DISPLAY</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>PIC X(8) USAGE DISPLAY</td>
<td>TID¹</td>
</tr>
</tbody>
</table>

Note: ¹ This data type is a CA IDMS extension of the SQL standard. For more information on documentation of CA IDMS data types, see Data Types (https://docops.ca.com/display/IDMSCU/Data+Types).

### 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:

<table>
<thead>
<tr>
<th>CA IDMS data type</th>
<th>Data type in included table</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY(n)</td>
<td>PIC X(n)</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>PIC X(n)</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>-LEN PIC S9(4) COMP</td>
</tr>
<tr>
<td></td>
<td>-TEXT PIC X(n)</td>
</tr>
<tr>
<td>GRAPHIC(n)</td>
<td>PIC G(n) DISPLAY-1</td>
</tr>
<tr>
<td>VARGRAPHIC(n)</td>
<td>-LEN PIC S9(4) COMP</td>
</tr>
<tr>
<td></td>
<td>-TEXT PIC G(n) DISPLAY-1</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>PIC S9(p-s)V9(s) COMP-3</td>
</tr>
<tr>
<td>UNSIGNED DECIMAL(p,s)</td>
<td>PIC 9(p-s)V9(s) COMP-3</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>PIC S9(p-s)V9(s) DISPLAY</td>
</tr>
</tbody>
</table>
CA IDMS data type | Data type in included table
---|---
UNSIGNED NUMERIC(p,s) | PIC 9(p-s)V9(s) DISPLAY
DOUBLE PRECISION | COMP-2
FLOAT(n) where  
\( n \leq 24 \) | COMP-1
\( n > 24 \) | 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
*Indicator variable* | PIC S9(4) COMP or PIC S9(8) COMP
TID | PIC X(8)

### Defining 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:

```plaintext
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.
```
Using SQL in a CA ADS Application

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

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 (:

Note: For more information on host variables, see Referring to Host Variables (see page 29).

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
```

**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:

**COBOL/CA ADS SQLCA**

```
01 SQLCA.
  02 SQLCAID PIC X(8).
  02 SQLCODE PIC S9(9) COMP.
  02 SQLCSID PIC X(8).
  02 SQLCINFO.
    03 SQLCERC PIC S9(9) COMP.
    03 FILLER PIC S9(9) COMP.
    03 SQLCNRP PIC S9(9) COMP.
    03 FILLER PIC S9(9) COMP.
    03 SQLCSER PIC S9(9) COMP.
    03 FILLER PIC S9(9) COMP.
    03 SQLCLNO PIC S9(9) COMP.
    03 SQLCMCT PIC S9(9) COMP.
    03 SQLCARC PIC S9(9) COMP.
    03 SQLCFJB PIC S9(9) COMP.
    03 FILLER PIC S9(9) COMP.
    03 FILLER PIC S9(9) COMP.
  02 SQLCINF2 REDEFINES SQLCINFO.
    03 SQLERRD PIC S9(9) COMP
      OCCURS 12.
  02 SQLCMSG.
    03 SQLCERL PIC S9(9) COMP.
    03 SQLERM PIC X(256).
  02 SQLCMSG2 REDEFINES SQLCMSG.
    03 FILLER PIC X(2).
    03 SQLERM.
```
Using SQL in a COBOL Application Program

Contents

- Embedding SQL Statements (see page 85)
  - Delimited, Continued, and Commented Statements (see page 85)
  - Placing an SQL Statement (see page 87)
- Defining Host Variables (see page 88)
  - Using COBOL Data Declarations (see page 88)
  - Using INCLUDE TABLE (see page 91)
  - Defining Bulk Structures (see page 92)
  - Non-bulk Structures and Indicator Arrays (see page 93)
- Referring to Host Variables (see page 96)
- Including SQL Communication Areas (see page 97)
- Copying Information from the Dictionary (see page 98)
- COPY IDMS FILE Statement (see page 99)
  - Syntax (see page 99)
  - Parameters (see page 99)
  - Usage (see page 99)
- COPY IDMS RECORD Statement (see page 99)
  - Syntax (see page 99)
  - Parameters (see page 99)
  - Usage (see page 100)
- COPY IDMS MODULE Statement (see page 101)
  - Syntax (see page 101)
  - Parameters (see page 101)
  - Usage (see page 101)
- INCLUDE Module-name Statement (see page 101)
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.

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 following example 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
           'TELEPHONE, 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 following example 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.
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 II

If your program is written for a version of COBOL that is prior to VS COBOL II, 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 II 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 II, 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.
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.

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 the SQL standard 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:

<table>
<thead>
<tr>
<th>COBOL PICTURE and USAGE clause</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC X(n) USAGE DISPLAY</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>01 name</td>
<td>VARCHARG(n)</td>
</tr>
<tr>
<td>49 name-LEN PIC S9(4) COMP</td>
<td></td>
</tr>
<tr>
<td>49 name-TEXT PIC X(n)</td>
<td></td>
</tr>
<tr>
<td>PIC S9(p-s)V9(s) USAGE COMP-3</td>
<td>DECIMAL(p,s)</td>
</tr>
<tr>
<td>PIC 9(p-s)V9(s) USAGE COMP-3</td>
<td>UNSIGNED DECIMAL(p,s)²</td>
</tr>
<tr>
<td>USAGE COMP-2</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>USAGE COMP-1</td>
<td>REAL</td>
</tr>
<tr>
<td>USAGE COMP-1</td>
<td>FLOAT</td>
</tr>
<tr>
<td>PIC S9(n) USAGE COMP (where n&lt;5)</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>PIC S9(n) USAGE COMP (where n&gt;4 and n&lt;10)</td>
<td>INTEGER</td>
</tr>
<tr>
<td></td>
<td>LONGINT or BIGINT</td>
</tr>
</tbody>
</table>
### COBOL PICTURE and USAGE clause

<table>
<thead>
<tr>
<th>COBOL PICTURE and USAGE clause</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9((n)) USAGE COMP</td>
<td></td>
</tr>
<tr>
<td>(where (n) &gt; 9)</td>
<td></td>
</tr>
<tr>
<td>PIC S9((p)-s)V9((s)) USAGE DISPLAY</td>
<td>NUMERIC((p),(s))</td>
</tr>
<tr>
<td>PIC 9((p)-s)V9((s)) USAGE DISPLAY</td>
<td>UNSIGNED NUMERIC((p),(s))</td>
</tr>
<tr>
<td>PIC X((n)) USAGE SQLBIN</td>
<td>BINARY((n))</td>
</tr>
<tr>
<td>PIC G((n)) USAGE DISPLAY-1</td>
<td>GRAPHIC((n))</td>
</tr>
<tr>
<td>01 name</td>
<td></td>
</tr>
<tr>
<td>49 name-LEN PIC S9(4) COMP</td>
<td></td>
</tr>
<tr>
<td>49 name-TEXT PIC G((n)) DISPLAY-1</td>
<td></td>
</tr>
<tr>
<td>PIC X(10) USAGE DISPLAY</td>
<td>DATE</td>
</tr>
<tr>
<td>PIC X(8) USAGE DISPLAY</td>
<td>TIME</td>
</tr>
<tr>
<td>PIC X(26) USAGE DISPLAY</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>PIC X(8) USAGE SQLBIN</td>
<td>TID(^1)</td>
</tr>
</tbody>
</table>

**Note:** \(^1\)This data type is a CA IDMS extension of the SQL standard. For more information on CA IDMS data types, see the **CA IDMS SQL Reference section**.

### 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:

```cobol
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:

```cobol
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-1
  - COMP-2
  - COMP-3
  - SQLIND
  - SQLBIN
  - SQLSESS
- VALUE clause
- 88 condition-name (any legal COBOL clause)
- OCCURS 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\(^1\) clause (except within a bulk or non-bulk structure)
- BLANK WHEN ZERO\(^1\) (except within a bulk or non-bulk structure)
- SYNCHRONIZED\(^2\) (except within a bulk or non-bulk structure)

\(^1\) This support is a CA IDMS extension of the SQL standard.

### 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.

<table>
<thead>
<tr>
<th>CA IDMS data type</th>
<th>COBOL data type on INCLUDE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY((n))</td>
<td>PIC X((n))</td>
</tr>
<tr>
<td>CHARACTER((n))</td>
<td>PIC X((n))</td>
</tr>
<tr>
<td>VARCHAR((n))</td>
<td>-LEN PIC S9(4) COMP</td>
</tr>
<tr>
<td></td>
<td>-TEXT PIC X((n))</td>
</tr>
<tr>
<td>GRAPHIC((n))</td>
<td>PIC G((n)) DISPLAY-1</td>
</tr>
<tr>
<td>VARGRAPHIC((n))</td>
<td>-LEN PIC S9(4) COMP</td>
</tr>
<tr>
<td></td>
<td>-TEXT PIC G((n)) DISPLAY-1</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>PIC S9((p-s))V9((s)) COMP-3</td>
</tr>
<tr>
<td>UNSIGNED DECIMAL(p,s)</td>
<td>PIC 9((p-s))V9((s)) COMP-3</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>PIC S9((p-s))V9((s)) DISPLAY</td>
</tr>
<tr>
<td>UNSIGNED NUMERIC(p,s)</td>
<td>PIC 9((p-s))V9((s)) DISPLAY</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>COMP-2</td>
</tr>
<tr>
<td>FLOAT((n)) where</td>
<td>COMP-1</td>
</tr>
<tr>
<td>(n \leq 24)</td>
<td>COMP-2</td>
</tr>
<tr>
<td>(n &gt; 24)</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>COMP-1</td>
</tr>
<tr>
<td>DATE</td>
<td>PIC X((10))</td>
</tr>
<tr>
<td>TIME</td>
<td>PIC X((8))</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>PIC X((26))</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>PIC S9((4)) COMP</td>
</tr>
<tr>
<td>INTEGER</td>
<td>PIC S9((8)) COMP</td>
</tr>
<tr>
<td>LONGINT</td>
<td>PIC S9((18)) COMP</td>
</tr>
<tr>
<td>SQLIND</td>
<td>COMP PIC S9((8))</td>
</tr>
<tr>
<td>TID</td>
<td>PIC X((8)) USAGE SQLBIN</td>
</tr>
</tbody>
</table>

### 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.

Note: For more information on precompiler options, see Preparing and Executing the Program (see page 114).

Defining 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.

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:

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

```sql
EXEC SQL
BEGIN DECLARE SECTION
  02 IND.
     04 IND SQLIND OCCURS 20 TIMES.
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:

  ```sql
  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:

```sql
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.

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 (:).

Note: For more information, see Referring to Host Variables (see page 29).

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:

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

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

Subscripted Variable Names
A CA IDMS extension of the SQL standard supports host variable arrays for use in bulk processing. By further extension of the SQL standard, 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)

### 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:
  
  ```cobol
  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:
  
  ```cobol
  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:
  
  ```sql
  EXEC SQL INCLUDE SQLCA END-EXEC.
  ```

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

**SQLCA Structure**

This is the COBOL format of the SQLCA:
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
COPY IDMS FILE Statement

The 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. Any records included in the file through the Integrated Data Dictionary (IDD) facilities are also copied.

Syntax

FILE SECTION.

  COPY IDMS FILE file-name VERSION version-number

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.

Usage

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

COPY IDMS RECORD Statement

The COPY IDMS RECORD statement 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.

Syntax

WORKING-STORAGE SECTION.
  LINKAGE SECTION.

  COPY IDMS RECORD record-specification
    level-number

Expansion of Record-specification

record-name VERSION version-number

REDEFINES record-name

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 9999.

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. Therefore, two record descriptions can provide alternative definitions of the same storage location.

**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.
COPY IDMS MODULE Statement

The COPY IDMS MODULE statement copies source statements from a module stored in the data dictionary into the source program.

Syntax

PROCEDURE DIVISION.

COPY IDMS module module-name

VERSION version-number

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.

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.

INCLUDE Module-name Statement

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

⚠️ **Note:** For more information on this statement, see the CA IDMS SQL Reference section.
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.

---

Note: For more information on non-SQL precompiler directives, see [Precompiler Directives](#) (see page 221).

---

Using SQL in a PL/I Application Program

Contents

- Embedding SQL Statements (see page 103)
  - Declaring SQLXQ1 (see page 103)
  - Delimited, Continued, and Commented Statements (see page 103)
- Defining Host Variables (see page 104)
  - Using PL/I Declarations (see page 105)
  - Using INCLUDE TABLE (see page 106)
  - Defining Bulk Structures (see page 108)
- Referring to Host Variables (see page 109)
- Including SQL Communication Areas (see page 110)
- Including Information from the Dictionary (see page 111)
- INCLUDE IDMS Record Statement (see page 111)
  - Syntax (see page 111)
  - Parameters (see page 112)
  - Usage (see page 112)
- INCLUDE IDMS MODULE statement (see page 112)
  - Syntax (see page 112)
  - Parameters (see page 113)
- INCLUDE Module-name Statement (see page 113)
- Non-SQL Precompiler Directives (see page 114)
This section presents information that is specific to embedding SQL in a PL/I application program.

Note: For more information on documentation of all aspects of PL/I application programming in the CA IDMS environment, see DML Reference for PL/I (https://docops.ca.com/pages/viewpage.action?pageId=309428803).

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.

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);
```

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 following example 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 following example 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') ;

**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
## 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:

<table>
<thead>
<tr>
<th>Equivalent PL/I data type</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR (n)</td>
<td>CHAR (n)</td>
</tr>
<tr>
<td>CHAR (n) VAR</td>
<td>VARCHAR (n)</td>
</tr>
<tr>
<td>FIXED DECIMAL (p,s)</td>
<td>DECIMAL (p,s)</td>
</tr>
<tr>
<td>FLOAT BINARY (n)</td>
<td>REAL</td>
</tr>
<tr>
<td>where n &lt;= 24</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>where n &gt; 24</td>
<td></td>
</tr>
<tr>
<td>FLOAT DECIMAL (n)</td>
<td>REAL</td>
</tr>
<tr>
<td>where n &lt;= 6</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>where n &gt; 6</td>
<td></td>
</tr>
<tr>
<td>FIXED BINARY (15)</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>FIXED BINARY (31)</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CHAR (n)</td>
<td>BINARY (n)</td>
</tr>
<tr>
<td>GRAPHIC (n)</td>
<td>GRAPHIC (n) ¹</td>
</tr>
<tr>
<td>GRAPHIC (n) VAR</td>
<td>VARGRAPHIC (n)</td>
</tr>
<tr>
<td>CHAR (10)</td>
<td>DATE</td>
</tr>
<tr>
<td>CHAR (8)</td>
<td>TIME</td>
</tr>
<tr>
<td>CHAR (26)</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>SQLBIN (n)</td>
<td>BINARY (n)</td>
</tr>
<tr>
<td>CHAR (8)</td>
<td>TID ¹</td>
</tr>
</tbody>
</table>

**Note:** ¹This data type is a CA IDMS extension of the SQL standard. For more information on CA IDMS data types, see the [Data Types](https://docops.ca.com/display/IDMSCU/Data+Types).

### Data Types Not Supported

The following table 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.

<table>
<thead>
<tr>
<th>Compatible PL/I data type</th>
<th>CA IDMS data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED BINARY (31)</td>
<td>LONGINT or BIGINT</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>CA IDMS data type</th>
<th>PL/I data type on INCLUDE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY(n)</td>
<td>CHAR (n)</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>CHAR (n)</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>CHAR (n) VAR</td>
</tr>
<tr>
<td>GRAPHIC(n)</td>
<td>GRAPHIC (n)</td>
</tr>
<tr>
<td>VARGRAPHIC(n)</td>
<td>GRAPHIC (n) VAR</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>FIXED DECIMAL (p,s)</td>
</tr>
<tr>
<td>UNSIGNED DECIMAL(p,s)</td>
<td>FIXED DECIMAL (p,s)</td>
</tr>
<tr>
<td>NUMERIC(p,s)(^1)</td>
<td>FIXED DECIMAL (p,s)</td>
</tr>
<tr>
<td>UNSIGNED NUMERIC(p,s)</td>
<td>FIXED DECIMAL (p,s)</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>FLOAT BINARY (53)</td>
</tr>
<tr>
<td>FLOAT (n) with n &lt;= 24</td>
<td>FLOAT BINARY (21)</td>
</tr>
<tr>
<td>FLOAT (n) with n &gt; 24</td>
<td>FLOAT BINARY (53)</td>
</tr>
<tr>
<td>REAL</td>
<td>FLOAT BINARY (21)</td>
</tr>
<tr>
<td>DATE</td>
<td>CHAR (10)</td>
</tr>
<tr>
<td>TIME</td>
<td>CHAR (8)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>CHAR (26)</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>FIXED BINARY (15)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>FIXED BINARY (31)</td>
</tr>
<tr>
<td>LONGINT</td>
<td>FIXED BINARY (31)</td>
</tr>
<tr>
<td>SQLIND</td>
<td>FIXED BINARY (31)</td>
</tr>
<tr>
<td>TID</td>
<td>CHAR (8)</td>
</tr>
</tbody>
</table>

### 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.

⚠️ **Note:** For more information on precompiler options, see [Precompiler Directives](#).
Defining 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:

```sql
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:

```sql
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;

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 (:).

Note: For more information, see Data Manipulation with SQL (see page 51).

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 the SQL standard supports host variable arrays for use in bulk processing. By further extension of the SQL standard, CA IDMS supports reference to a subscripted variable in a host variable array.

All of the following are valid host variable references:
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:

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

- The program can declare the host variable SQLCODE:

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

- The program can issue this directive:

```pli
EXEC SQL INCLUDE SQLCA ;
```

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

SQLCA Structure

This is the PL/I format of the SQLCA:

**PL/I SQLCA**

```pli
DECLARE 1 SQLCA,
  2 SQLCAID CHARACTER (8),
  2 SQLCODE FIXED BINARY (31),
  2 SQLCSID CHARACTER (8),
  2 SQLCINFO,
  3 SQLCERC FIXED BINARY (31),
  3 FILLER
  3 SQLCMCT FIXED BINARY (31),
  3 SQLCARC FIXED BINARY (31),
  3 SQLCFJB FIXED BINARY (31),
  3 FILLER
  3 SQLCARC FIXED BINARY (31),
  3 FILLER
  3 SQLCMCT FIXED BINARY (31),
  3 SQLCARC FIXED BINARY (31),
  3 SQLCFJB FIXED BINARY (31),
  3 FILLER
  3 SQLCARC FIXED BINARY (31),
  3 FILLER
  3 SQLCMCT FIXED BINARY (31),
  3 SQLCARC FIXED BINARY (31),
  3 SQLCFJB FIXED BINARY (31),
  3 FILLER
  3 SQLCARC FIXED BINARY (31),
  3 FILLER
  3 SQLCMCT FIXED BINARY (31),
  3 SQLCARC FIXED BINARY (31),
  3 SQLCFJB FIXED BINARY (31),
  3 FILLER
```
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

INCLUDE IDMS Record Statement

The INCLUDE IDMS Record statement is used to copy record descriptions into the program and can be coded in your application program.

Syntax

```plaintext
; INCLUDE IDMS record-specification
  level-number
```

Expansion of Record Specification

```plaintext
record-name
  VERSION version-number

attribute
```
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*.

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**

**INCLUDE IDMS MODULE statement**

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

**Syntax**

```
INCLUDE IDMS (module-name);  VERSION version-number
```
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.

  **Note:** For more information on the DDDL compiler, see the IDD DDDL Reference (https://docops.ca.com/display/IDMSCU/IDD+DDDL+Reference).

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. 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.

**INCLUDE Module-name Statement**

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

**Note:** For more information on this statement, see INCLUDE (https://docops.ca.com/display/IDMSCU/INCLUDE).
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.

**Note:** For more information on non-SQL precompiler directives, see Precompiler Directives (see page 221).

Preparing and Executing the Program

To put your source program into executable form, take the following steps:

- Precompiling the Program (see page 114)
- Compiling the Program (see page 119)
- Creating the Access Module (see page 120)
- Executing an SQL Application (see page 125)
- Testing the Access Module (see page 127)
- Debugging the Application (see page 128)

If you are using CA ADS, the CA ADS compiler ADSC performs steps 1 and 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.

- About the Precompiler (see page 115)
- Precompiler Options (see page 116)
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.

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.

⚠️ Note: For more information and documentation about the schema-name mapping for tables and views, see Creating the Access Module (see page 120).

How You Precompile

You precompile the program by submitting a batch job.

For precompiler JCL, see Sample JCL (see page 169).

You can specify parameters in the precompiler JCL that determine how the precompiler executes.

For documentation of precompiler parameters, see Precompiler Options (see page 116).

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

### Precompiler Options

#### Syntax

This is the expansion of `precompiler-options` in the precompiler EXEC PGM statement in JCL. These are not positional parameters:

- `RCM = rcm-name`
- `RCMVERSION = rcm-version-number`
- `AM = access-module-name`
- `SCHEMA = schema-name`
- `NOINSTALL`
- `DICTNAME = dictionary-name`
- `SQL = NO 89 FIPS`
- `LIST NOList`
- `DATE = ISO USA EUR JIS`
- `TIME = ISO USA EUR JIS`

**COBOL precompiler only**

`COBOL = 1`

2 85

#### 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.

  ! Note: For more information on sample precompiler JCL, see Sample JCL (https://docops.ca.com/display/IDMS19/SampleJCL).

  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**
  Specifies that compliance with a named SQL standard is not checked or enforced, and all CA IDMS
  extensions are permitted.
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.

  **Note:** For more information on NODMLIST, see Precompiler Directives (see page 221).

- **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.

  **Note:** For more information on DMLIST, see Precompiler Directives (see page 221).

- **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.

- **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.
### 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.
011600*    MOVE 4 TO SQLCLNO
011700*    MOVE 16 TO SQLCMD
011800*    MOVE 1 TO SQLARG
011900*    MOVE 4 TO SQLSID
011900*    MOVE 278 TO SQLTBL
011900*    MOVE 6 TO SQLMRO
011900*    MOVE INDEX-CNTR TO SQLSRO
011900*    MOVE NUM-ROWS TO SQLNR0
011900*    CALL 'IDMSSQL' USING
011900*        SQLRPB
011900*        SQLCA
011900*        SQLCA
011900*        SQLCIB
011900*        SQLPIB
011900*        SQLCA
011900*        EMPDATA
011900*        SQLCA
011900*        SQLCA
011900*        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.

Note: For JCL and more information on compiling and link editing a program see Sample JCL (see page 169).

Creating the Access Module

- Overriding Access Module Defaults (see page 121)
- Altering an Access Module (see page 125)
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

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.

⚠️ Note: For more information and specific rules regarding schema ownership and authority to execute access modules under CA IDMS security, see the Security Checking (https://docops.ca.com/display/IDMS19/Security+Checking).
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.

⚠️ **Note:** For more information, see Executing the Application (https://docops.ca.com/display/IDMS19/Executing+the+Application).

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 will change to the EMP_TSTSCH.EMPLOYEE table:

```sql
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,
  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.

⚠️ **Note:** For more information on altering an access module, see Altering an Access Module (see page 125).

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.

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. For more information, see the SET TRANSACTION (https://docops.ca.com/display/IDMSCU/SET+TRANSACTION).

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.

⚠️ Note: For more information on the effect of isolation level, see Writing an SQL Program (see page 24).

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.

⚠️ **Note:** For more information on ready mode options, see [CREATE ACCESS MODULE](https://docops.ca.com/display/IDMSCU/CREATE+ACCESS+MODULE) or [Ready Areas](https://docops.ca.com/display/IDMS19/Readying+Areas).

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:

<table>
<thead>
<tr>
<th>Specified ready mode</th>
<th>Area status</th>
<th>Actual ready mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(No specification)</td>
<td>Transient retrieval</td>
<td>Transient retrieval</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>Shared retrieval</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Shared retrieval</td>
</tr>
<tr>
<td>Any retrieval mode</td>
<td>Transient retrieval</td>
<td>Transient retrieval</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>As specified</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Shared retrieval</td>
</tr>
<tr>
<td>Any update mode</td>
<td>Transient retrieval</td>
<td>Transient retrieval</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>Shared retrieval</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Shared retrieval</td>
</tr>
</tbody>
</table>

**READ WRITE Ready Modes**

This table presents the actual ready modes when the transaction state is READ WRITE:

<table>
<thead>
<tr>
<th>Specified ready mode</th>
<th>Area status</th>
<th>Actual ready mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(No specification)</td>
<td>Transient retrieval</td>
<td>Transient retrieval</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>Shared retrieval</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>Shared update</td>
</tr>
<tr>
<td>Any retrieval mode</td>
<td>Transient retrieval</td>
<td>Transient retrieval</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>As specified</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>As specified</td>
</tr>
<tr>
<td>Any update mode</td>
<td>Transient retrieval</td>
<td>(Runtime error)</td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td>(Runtime error)</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>As specified</td>
</tr>
</tbody>
</table>
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.

⚠️ Note: For more information on altering an access module, see ALTER ACCESS MODULE (https://docops.ca.com/display/IDMSCU/ALTER+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:

```sql
EXEC SQL
    ALTER ACCESS MODULE EMPINFO1
    REPLACE ALL
    MAP EMP_SCH TO EMP_PRODSCH,
    MAP NULL TO EMP_PRODSCH
END-EXEC.
```

Executing an SQL 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 Sample JCL (see page 169).

SYSIDMS Parameters

In batch JCL, you can tailor certain aspects of the runtime environment by specifying SYSIDMS parameters. The following table lists the options specific to SQL processing:

<table>
<thead>
<tr>
<th>SYSIDMS parameter</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLTRACE</td>
<td>Activates or deactivates the facility that traces all SQL requests made by the application</td>
</tr>
<tr>
<td>PROCTRA CE=ON</td>
<td>ON activates a trace of key user blocks that participate in an SQL PROCEDURE call. OFF is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SYSIDMS parameter

SQL_CACH \( n \) specifies the max number of entries that will be used in the dynamic SQL cache. One entry holds one cached SQL statement. With \( n \) set to 0, dynamic SQL caching will be disabled. The theoretical max value for \( n \) is 2,147,483,647, but the real maximum is determined by available address space. The default is 200.

SQL_INTLS ORT=ON /OFF

Allows you to force the internal IDMS sort to be used in local mode. If ON is specified, an internal SORT rather than an operating system SORT will be performed on SQL commands issued in a local batch job that contains an ORDER BY clause. In many cases, an internal SORT is faster than an operating system SORT when you are not dealing with a large amount of data. OFF is the default, indicating an operating system SORT will be used.

**Note:** For more information and the complete list of available SYSIDMS parameters, see [SYSIDMS Parameter Descriptions](https://docops.ca.com/display/IDMS19/SYSIDMS+Parameter+Descriptions).

---

**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.

**Note:** For more information on qualifying the access module name, see [Overriding Access Module Defaults (see page 121)](https://docops.ca.com/display/IDMS19/Overriding+Access+Module+Defaults).

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.

**Note:** For more information on privileges required to access CA IDMS, see your security administrator.
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.

Note: For more information on associating a program with an access module, see Precompiling the Program (see page 114).

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

  Note: For more information on using the SET ACCESS MODULE statement, see Preparing and Executing the Program (see page 114) or SET ACCESS MODULE (https://docops.ca.com/display/IDMSCU/SET+ACCESS+MODULE).

- 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.

Note: For more information on test versions, see documentation of DCUF TEST (https://docops.ca.com/display/IDMSCU/DCUF+TEST).
Debugging the Application

CA IDMS provides these tools that you can use to debug the SQL portion of the application program:

- Command Facility (see page 128)
- SQL Trace Facility (see page 129)
- EXPLAIN Statement (see page 130)
- Online Debugger (see page 130)

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.

⚠️ **Note:** You can use CA OLQ to access CA IDMS with SQL. For more information, see the Online Query Reference (https://docops.ca.com/display/IDMSCU/Online+Query+Reference).

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 nn.n ONLINE IDMS NO ERRORS 1/16
SELECT PROJ_ID, EST_START_DATE, PROJ_DESC
FROM DEMOPROJ.PROJECT
WHERE EST_START_DATE > 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
```

⚠️ **Note:** For more information on using the Command Facility, see the Command Facility (https://docops.ca.com/display/IDMS19/Command+Facility).
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.

```sql
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 <= ' ';
```

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.

⚠️ **Note:** For more information on the requirements for calling IDMSIN01 to activate or deactivate the SQL trace facility, see [IDMSIN01](https://docops.ca.com/display/IDMSCU/IDMSIN01).
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 formulation 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.

⚠️ **Note:** For more information on the EXPLAIN statement and its use, see [EXPLAIN](https://docops.ca.com/display/IDMSCU/EXPLAIN).

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

⚠️ **Note:** For more information on debugging online application programs, see the [Administrating Online Debugger](https://docops.ca.com/display/IDMS19/Administrating+Online+Debugger).

SQL Programming Techniques

Programming techniques that increase the processing capability of the program and reduce the demand for system resources are necessary for optimum performance. In several cases, you can achieve these results because of CA IDMS SQL extensions.

- Modularized Programming (see page 130)
- Pseudoconversational Programming (see page 135)
- Managing Concurrent Sessions (see page 140)
- Creating and Using a Temporary Table (see page 143)
- Bill-of-materials Explosion (see page 146)

Modularized Programming
You can design an SQL application using modularized programming techniques. CA IDMS provides extensions to the SQL standard 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

Sharing a Cursor

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:

```sql
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 the following example, program EMPUPD declares an external cursor to share the global cursor declared in EMPGET:

```sql
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 the responsibility of 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.

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-NULL-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.

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.

Note: For information on how an access module is associated with a program, see Preparing and Executing the Program (see page 114).
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.

Note: For more information and a list of statements that can precede SET ACCESS MODULE in a database transaction, see the SET ACCESS MODULE (https://docops.ca.com/display/IDMSCU/SET+ACCESS+MODULE).

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.

Note: 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-STOREAGE 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
Pseudoconversational Programming

- Using SUSPEND SESSION and RESUME SESSION (see page 135)
- Scrolling Through a List of Rows (see page 136)
- Updating a Row After a Pseudoconverse (see page 137)

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.

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 discuss how to use SUSPEND SESSION and RESUME SESSION in these types of processing.

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:

```sql
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:

```sql
EXEC SQL
  FETCH POSITION CRSR
  BULK :BULK-POSITION
END-EXEC.
```

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

```sql
EXEC SQL
  SUSPEND SESSION
END-EXEC.
```

**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

⚠️ **Note:** For more information on scratch area management, see the applicable CA IDMS program language reference manual.

### 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. Therefore, 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:

```sql
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.
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.

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,
```
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:

```sql
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.
```

Managing Concurrent Sessions

- Session Management Concepts (see page 141)
- Implementing Concurrent Sessions (see page 141)

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 When Transaction Sharing Is Not in Effect

If an application attempts to access the same database in concurrent sessions, there is an inherent risk of deadlock; however, transaction sharing can be used to avoid such deadlocks.

Note: For more information on the use of transaction sharing, see Writing an SQL Program (see page 24) and the Transaction Sharing (https://docops.ca.com/display/IDMSCU/Transaction+Sharing).
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.

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.
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:

```
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.

Mainline

```
Connect to DB01
Save first session value
Initialize SQLSESS
Connect to DB02
Save second session value
Move first session value to SQLSESS
Call PROGRAM1 passing SQLSESS ...```

Program1

```
LINKAGE SECTION.
EXEC SQL
BEGIN DECLARE SECTION
END-EXEC.
01 SQLSESS USAGE SQLSESS
EXEC SQL
END DECLARE SECTION
END-EXEC.
...```
SQL statements for DB01

Move second session value to SQLSESS
Call PROGRAM2 passing SQLSESS

Program2

LINKAGE SECTION.
EXEC SQL
BEGIN DECLARE SECTION
END-EXEC.
01 SQLSESS USAGE SQLSESS
EXEC SQL
END DECLARE SECTION
END-EXEC.

FINISH TASK

SQL statements for DB02

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.

Mainline

Connect to DB01
Call PROGRAM1

Program1

Connect to DB02
Retrieve data
COMMIT RELEASE

Update data in DB01
COMMIT RELEASE

Creating and Using a Temporary Table

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 on the physical characteristics of a temporary table.

Note: For more information on creating temporary tables in particular, see CREATE TEMPORARY TABLE (https://docops.ca.com/display/IDMSCU/CREATE+TEMPORARY+TABLE). For more information on creating tables in general, see CREATE TABLE (https://docops.ca.com/display/IDMSCU/CREATE+TABLE).

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.

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:

```sql
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:

```sql
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:

```sql
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:

```sql
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.
```

**Bill-of-materials Explosion**

- What to Do (see page 146)
- Sample Program (see page 148)

This section presents a sample program that performs a bill-of-materials explosion. A discussion of the concepts involved precedes the sample program.

**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.

```sql
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.

```sql
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
```

```sql
```
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:

```plaintext
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.

```plaintext
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. The bill-of-materials explosion is now complete.

### Sample Program

```plaintext
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.

*********************************************************
***** DECREASE 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 = 'O'
    MOVE 'O' 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.
  END-IF.
CLOSE-CURRENT-CURSOR SECTION.
  IF CURSOR-FLAG (CURRENT-CURSOR) = 'O'
    MOVE ' ' TO 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.
  END-IF.

BACKUP-ONE-LEVEL SECTION.
  SUBTRACT 1 FROM CURRENT-LEVEL.
  IF CURRENT-LEVEL > 0
    MOVE SAVE-KEY (CURRENT-LEVEL) TO PREVIOUS-COMPONENT.
  END-IF.
  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.
    END-IF.
PRINT-CURRENT-ROW SECTION.
  MOVE CURRENT-LEVEL TO DISPLAY-LEVEL.
  IF CURRENT-LEVEL > MAX-LEVELS
    MOVE 'MAXIMUM LEVEL, COMPONENTS NOT LISTED'
Dynamic SQL

Depending on the processing requirement of the program and the capabilities of the programming language, you will need to implement dynamic SQL.

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.

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 SQL Invoked Procedure

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, Local Variables, or Routine Parameters

A dynamic SQL statement that is prepared or executed using an EXECUTE IMMEDIATE statement cannot reference host variables, local variables, or routine parameters 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 variables or parameters.

⚠️ Note: For more information on dynamic parameters, see the Dynamic Parameters (https://docops.ca.com/display/IDMSCU/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 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. As general practice, you should avoid specifying the NOINSTALL option.

Dynamic Insert, Update, and Delete Operations

Contents

- Using EXECUTE IMMEDIATE (see page 154)
- Using PREPARE (see page 155)
- Using EXECUTE (see page 157)

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.

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).
```
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.

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 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

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 on 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:

```sql
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 Using section. This work record must match the SQLDA layout and the initial values should conform to the data types.

The following example displays the CA ADS format of the SQLDA:

```sql
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 on 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 on a result column.
PREPARE Example

In this example, the program has formulated an SQL statement and has moved the character string into the host variable STATEMENT-STRING:

```sql
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.

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:

```sql
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:

```sql
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.

**Executing Prepared SELECT Statements**

- **What to Do (see page 158)**
- **Sample Program (see page 159)**

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.

**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:

```sql
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:

```sql
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:

```sql
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
```
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:

```
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.
```

Sample Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EMPVIEW1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 SQLMSGS.
     02 SQLMAX         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).
EXEC SQL  BEGIN DECLARE SECTION      END-EXEC
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.
Executing Prepared CALL Statements

- What to Do (see page 162)
  - Declaring a Cursor (see page 162)
  - Preparing the Statement (see page 162)
  - Building the Statement Text (see page 162)
  - Declaring Host Variables for 3 Parameters (see page 163)
  - Sample Program (see page 163)

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.

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 the 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.

### Sample Program

IDENTIFICATION DIVISION.
PROGRAM-ID. DYNCALL.
  *-------------------------------------------------------------------*
  *                                                                  *
  * DYNCALL will read a procedure-reference and execute it dynamically.*
  *                                                                  *
  * It is assumed that the procedure has 3 parameters, P1 and P2 are numeric, P3 is alphanumeric.*
  *-------------------------------------------------------------------*
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 SQLMSGS.
   02 SQLMMAX  PIC S9(8) COMP VALUE +6.
   02 SQLSIZE  PIC S9(8) COMP VALUE +80. *
   02 SQLMCNT  PIC S9(8) COMP.  
   02 SQLMLINE OCCURS 6 TIMES PIC X(80).
  01 REQ-WK.
EXEC SQL BEGIN DECLARE SECTION
END-EXEC

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.
ELSE
  MOVE INPUT-LINE TO TEXT-LINES(TEXT-CTR).
ENDIF-STATEMENT

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.
ENDIF-STATEMENT

DISP-MSG SECTION.
DISPLAY SQLMLINE (LINE-CNT).
Dynamic SQL Caching

Contents

- Searching the Cache (see page 166)
- Impact of Database Definition Changes (see page 167)
  - SQL-Defined Databases and Caching (see page 167)
  - Non-SQL-Defined Databases and Caching (see page 168)
- Controlling the Cache (see page 168)

Dynamic SQL caching is a common technique used to improve performance in an SQL environment. Caching works in the following manner: when a dynamic SQL statement is compiled, a copy of the SQL statement and the result of the SQL compilation are saved in a cache. For each subsequent SQL compilation request, the cache is searched. If the statement is found, the matching compiled structures are used instead of recompiling the statement. This improves performance by eliminating the I/O requests to read the catalog and the CPU usage required to invoke the SQL optimizer for subsequent executions of the same dynamic SQL statement.

In most cases, the savings in resource consumption due to bypassing the SQL compilation are significantly greater than the extra cost associated with caching the SQL source, access plans, and related structures.

Note: At this time, only the SELECT, UPDATE, and DELETE SQL statements are cacheable.

Searching the Cache

When a search is made in the cache for a matching SQL statement, a cache hit occurs when a matching entry is found. The following factors are considered in determining whether an SQL statement matches a cache entry:

- The text of the statement
- The default schema in effect for the SQL session
- The dictionary to which the SQL session is connected
- Whether the statement references temporary tables

A literal comparison of the statement's text is made against each cache entry until a match is found. A literal comparison avoids the overhead of parsing but has the consequence that an entry may not match because of differences in such things as case and spacing. For example, the following three statements are considered different if using a literal comparison:
Select * from EMPLOYEE
Select * from EMPLOYEE
select * from employee

Specifying values as literals instead of as dynamic parameters can also result in unequal comparisons. The following two statements would be textually identical if a dynamic parameter had been used in place of the numeric values 100 and 101:

select * from DEMEMPL.EMPLOYEE where EMP_ID = 100
select * from DEMEMPL.EMPLOYEE where EMP_ID = 101

⚠️ Note: While the use of dynamic parameters can increase the frequency of finding a matching cache entry, it may occasionally prevent the optimizer from choosing the most efficient access strategy.

When a dynamic statement that relies on a default schema is cached, both the statement text and the default schema are saved. When the cache is searched for a statement that relies on a default schema, both the statement's text and the session's default schema must be equal to their cached equivalents for the entry to match. Consider the following two statements. The first will match a cached entry regardless of the default schema in effect for the SQL session. The second will match only if the default schema in effect for the SQL session is the same as that in the cache:

select * from DEMEMPL.EMPLOYEE
select * from EMPLOYEE

The name of the dictionary to which an SQL session is connected is always saved in the cache and compared to the session's dictionary during a search of the cache. If the two are not the same, then the cache entry does not match.

If an SQL statement references a temporary table, it will not be cached since each temporary table instance can be structurally different from others of the same name. Therefore, no statement that references a temporary table will match a cache entry.

Impact of Database Definition Changes

Database definition changes may or may not be detected automatically based on whether the database is SQL-defined or non-SQL-defined. This has consequences for dynamic SQL caching as explained next.

SQL-Defined Databases and Caching

Because SQL-defined databases have an associated catalog and because areas for SQL-defined databases have timestamps, CA IDMS is able to automatically detect definitional changes that impact cached SQL statements. Whenever a statement needs recompilation, CA IDMS automatically detects this condition and recompiles the affected statement dynamically.
Non-SQL-Defined Databases and Caching

Non-SQL-defined databases do not have timestamps for automatically determining whether a database's definition accurately describes the underlying data. Consequently, when changing the structure of a non-SQL-defined database, it is the administrator's responsibility to ensure that all SQL statements impacted by the change are recompiled. If dynamic SQL caching is not used, then this entails recompiling access modules that reference the affected database. If dynamic SQL caching is used, then it also entails purging the cache of statements that reference the affected database. This can be done by deleting rows from the SYSCA.DSCCACHE or SYSCA.DSCCACHEV tables.

**Note:** For more information on these tables, see the [Tables for Viewing, Monitoring, and Controlling the Cache](https://docops.ca.com/display/IDMSCU/Tables+for+Viewing%2C+Monitoring%2C+and+Controlling+the+Cache).

It is also recommended that dynamic SQL caching be disabled during the transition period in which the definitional changes are being implemented. For information on how to do this, see *Controlling the Cache* (see page 168).

CA IDMS will detect the need to recompile cached SQL statements if a change is made to the referencing SQL schema through which a non-SQL-defined schema is referenced. It does this by comparing the update stamp of the referencing SQL schema to the compile stamp of the cached statement.

Controlling the Cache

There are various ways that an individual user and a DBA can control dynamic SQL caching. Three ways are discussed following:

- Establishing caching attributes for an individual SQL session by issuing a SET SESSION statement
- Establishing default caching attributes for a central version through a system generation SQL CACHE statement
- Establishing default caching attributes for a local mode job by specifying a SYSIDMS SQL_CACHE_ENTRIES parameter.

**Note:** For more information on the SET SESSION statement, see [SET SESSION](https://docops.ca.com/display/IDMSCU/SET+SESSION). For more information on the various tables that control caching and examples of how to display and control the cache using SQL, see the [SQL Cache Tables](https://docops.ca.com/display/IDMSCU/SQL+Cache+Tables). For more information on the SQL CACHE system generation statement, see the [SQL CACHE Statement -- Controls SQL Caching](https://docops.ca.com/display/IDMS19/SQLCACHE+Statement++Controls+SQL+Caching). For more information on SYSIDMS parameter SQL_CACHE_ENTRIES, see [SYSIDMS Parameter Descriptions](https://docops.ca.com/display/IDMS19/SYSIDMS+Parameter+Descriptions).
Sample JCLs

Sample JCL or commands for executing the precompile, access module creation, compile, and link edit steps on four operating systems are provided in this section.

- z/OS (see page 169)
- z/VSE (see page 173)
- z/VM (see page 175)

z/OS

The following sample JCL streams contain 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

```cl
//*************************************************************
//**             PRECOMPILE COBOL PROGRAM                        **
//*************************************************************
//precomp EXEC PGM=IDMSDMLC,REGION=1024K,
//  PARM='optional precompiler parameters'
//STEPLIB DD DSN=idsms.dba.loadlib,DISP=SHR
//  DSN=idsms.loadlib,DISP=SHR
//sysctl DD DSN=idsms.sysctl,DISP=SHR
//dcmmsg DD DSN=idsms.sysmsg.dldcmsg,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))
//SYS004 DD UNIT=disk,SPACE=(TRK,(10,10))
//SYS005 DD UNIT=disk,SPACE=(TRK,(10,10))
//SYS006 DD UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//  DCB=(RECFM=FB,LRECL=80,BLKSZ=3120)
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
//DMCL=dmcl-name
//DICTNAME=dictionary-name
//Additional SYSIDMS parameters, as appropriate/*
//SYSIPT DD *
//Host language source statements with embedded SQL/
//*************************************************************
//*************************************************************
//**             CREATE ACCESS MODULE                          **
//*************************************************************
//acccmod EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB DD DSN=idsms.dba.loadlib,DISP=SHR
//  DSN=idsms.loadlib,DISP=SHR
//sysctl DD DSN=idsms.sysctl,DISP=SHR
//dcmmsg DD DSN=idsms.sysmsg.dldcmsg,DISP=SHR
```
Local Mode JCL
CA IDMS - 19.0

Host language source statements with embedded SQL

CREATE ACCESS MODULE

EXEC PGM=IDMSBCF,REGION=1024K
//accmod EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DSN=idms.loadlib,DISP=SHR
//dictb DD DSN=idms.appdict.dlldml,DISP=SHR
//dloddb DD DSN=idms.appdict.dlldclod,DISP=SHR
//sqladd DD DSN=idms.syssql.ddlcat,DISP=SHR
//sqlxadd DD DSN=idms.syssql.ddlcattx,DISP=SHR
//sqladd DD DSN=idms.syssql.ddlcattl,DISP=SHR
//dcmsg DD DSN=idms.sysmsg.ddlcmd,DISP=SHR
//sysjrnl DD DSN=idms.tapejrnl,DISP=(NEW,CATLG,UNIT=tape//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 ;
/*
CREATE ACCESS MODULE statement
COMMIT WORK RELEASE ;
*/

COMPILE COBOL PROGRAM

EXEC PGM=IKFCBL00,REGION=240K,
//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)
/*
SYSPUNCH DD SYSOUT=A
SYSLIN DD DSN=.&.object,DISP=(OLD,DELETE)
*/

LINK PROGRAM MODULE

EXEC PGM=IEWL,REGION=300K,PARM='LET,LIST,XREF'
//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=usrer.loadlib,DISP=SHR
//SYSLIN DD DSN=.&.object,DISP=(OLD,DELETE)
// SYSMOD DD *
INCLUDE loadlib(IDMS) Non-CICS only
INCLUDE loadlib(IDMSCINT) CICS only
ENTRY userentry
NAME userprog(R)
/*
*/

Note: The link of CICS application programs that use IDMSCINT must incorporate JCL to resolve external reference DFHEI1. The particular JCL depends on the nature and language of your application. For more information, see the appropriate IBM CICS application programming documentation.
### Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>accmod</td>
<td>Stepname for batch Command Facility execution of the CREATE ACCESS MODULE statement</td>
</tr>
<tr>
<td>compile</td>
<td>Stepname for the compile step</td>
</tr>
<tr>
<td>dcmmsg</td>
<td>DDname of the system message area (DDLDCMSG)</td>
</tr>
<tr>
<td>dictb</td>
<td>DDname of the application dictionary definition area (DDLDML)</td>
</tr>
<tr>
<td>dictionary-name</td>
<td>Name of the dictionary containing the SQL definition areas</td>
</tr>
<tr>
<td>disk</td>
<td>Symbolic device name for workfiles</td>
</tr>
<tr>
<td>dloddb</td>
<td>DDname of the application dictionary definition load area (DDLDCLOD)</td>
</tr>
<tr>
<td>dmcl-name</td>
<td>Name of the DMCL</td>
</tr>
<tr>
<td>idms.appldict.ddldclod</td>
<td>Data set name of the application dictionary definition load area (DDLDCLOD)</td>
</tr>
<tr>
<td>idms.appldict.ddldml</td>
<td>Data set name of the application dictionary definition area (DDLDML)</td>
</tr>
<tr>
<td>idms.dba.loadlib</td>
<td>Data set name of the load library containing the DMCL and database name table load modules</td>
</tr>
<tr>
<td>idms.loadlib</td>
<td>Data set name of the load library containing the CA IDMS executable modules</td>
</tr>
<tr>
<td>idms.sysctl</td>
<td>Data set name of the SYSCTL file</td>
</tr>
<tr>
<td>idms.sysmsg.ddlcmmsg</td>
<td>Data set name of the system message area (DDLDCMSG)</td>
</tr>
<tr>
<td>idms.syssql.ddlcat</td>
<td>Data set name of the SQL definition area (DDLCAT) of the application dictionary</td>
</tr>
<tr>
<td>idms.syssql.ddlcatl</td>
<td>Data set name of the SQL definition load area (DDLCATLOD) of the application dictionary</td>
</tr>
<tr>
<td>idms.syssql.ddlcatx</td>
<td>Data set name of the SQL definition index area (DDLCATX) of the application dictionary</td>
</tr>
<tr>
<td>idms.tapejrnl</td>
<td>Data set name of the tape journal file</td>
</tr>
<tr>
<td>loadlib</td>
<td>DDname of the load library containing the CA IDMS executable modules</td>
</tr>
<tr>
<td>precomp</td>
<td>Stepname for the precompile step</td>
</tr>
<tr>
<td>sqlidd</td>
<td>DDname of the SQL definition area (DDLCAT) of the application dictionary</td>
</tr>
<tr>
<td>sqllld</td>
<td>DDname of the SQL definition load area (DDLCATLOD) of the application dictionary</td>
</tr>
<tr>
<td>sqlxdd</td>
<td>DDname of the SQL definition index area (DDLCATX) of the application dictionary</td>
</tr>
<tr>
<td>syslog</td>
<td>DDname of the SYSCTL file</td>
</tr>
<tr>
<td>sysjrnl</td>
<td>DDname of the tape journal file</td>
</tr>
<tr>
<td>sys1.cobol.linklib</td>
<td>Data set name of the library containing the host language compiler module</td>
</tr>
<tr>
<td>sys1.coblib</td>
<td>Data set name of the library containing host language compiler subroutines</td>
</tr>
<tr>
<td>tape</td>
<td>Symbolic device name for tape journal file</td>
</tr>
<tr>
<td>userentry</td>
<td>Entry point for the user program</td>
</tr>
</tbody>
</table>
Variable | Definition
---|---
user.loadlib | Data set name of the load library containing executable modules for user programs

userprog | Name of the user program

&.&object. | Host language compiler output to be passed to the linkage editor

&.&source. | Precompiler output to be passed to the host language compiler

---

**z/VSE**

The following sample JCL stream 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 is a table that gives the meaning of variables used in the examples along with a set of usage notes.

---

**Central Version JCL**

```
*****************************************************************
** PREPARAMETER COBOL PROGRAM                                    **
*****************************************************************
// EXEC PROC=IDMSLBLS
// DLBL idmspch.temp.dmlc,0
// EXTENT SYS020,nnnnnn,,ssss,llll
// ASSIGN SYS020,DISK,VOL=nnnnnn,SHR
// EXEC IDMSDMLC
Optional precompiler parameters
/*
DMCL=dmcl-name
DICTNAME=dictionary-name
Additional SYSIDMS parameters, as appropriate
/*
Host language source statements with embedded SQL
/*
*****************************************************************
** CREATE ACCESS MODULE                                          **
*****************************************************************
// EXEC IDMSBCF
DMCL=dmcl-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
// ASSIGN SYSIPT,DISK,VOL=nnnnnn,SHR
// OPTION CATAL,NOECK,NSYM
```
Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary-name</td>
<td>Name of the dictionary containing the SQL definitions</td>
</tr>
<tr>
<td>dmcl-name</td>
<td>Name of the DMCL</td>
</tr>
<tr>
<td>f</td>
<td>File number of the tape journal file</td>
</tr>
<tr>
<td>idmspch</td>
<td>Host language compiler output to be passed to the linkage editor</td>
</tr>
<tr>
<td>idms.tapejrnl</td>
<td>File ID of the tape journal file</td>
</tr>
<tr>
<td>llll</td>
<td>Number of tracks (CKD) or blocks (FBA) of disk extent</td>
</tr>
<tr>
<td>nnnnnn</td>
<td>Volume serial identifier of appropriate disk volume</td>
</tr>
<tr>
<td>ssss</td>
<td>Starting track (CKD) or block (FBA) of disk extent</td>
</tr>
<tr>
<td>sysjrnl</td>
<td>Filename of the tape journal file</td>
</tr>
<tr>
<td>temp.dmlc</td>
<td>File ID of the precompiler output</td>
</tr>
<tr>
<td>userentry</td>
<td>Entry point for the user program</td>
</tr>
<tr>
<td>userprog</td>
<td>Name of the user program</td>
</tr>
</tbody>
</table>

Local Mode JCL

To execute in local mode, add these statements to the precompile step:

```c
// TLBL   sysjrnl,'idms.tapejrnl',nnnnnn,,f
// ASSGN  SYS009,TAPE,VOL=nnnnnn
```

⚠️ **Note:** The link of CICS application programs that use IDMSCINT must incorporate JCL to resolve external reference DFHEI1. The particular JCL depends on the nature and language of your application. See the appropriate IBM CICS application programming documentation for details.
Usage

IDMSLBLS Procedure

IDMSLBLS is a procedure provided during a CA IDMS z/VSE 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 z/VSE 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

z/VM

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 is a table that gives the meaning of variables used in the examples and a set of usage notes.

### Commands for Central Version Execution

```plaintext
/* **********************************************************
** PRECOMPILE COBOL PROGRAM                             **
/* **********************************************************/
FILEDEF sysipt1 DISK program source a
FILEDEF sysidms1 DISK sysidms1 parms a
FILEDEF syspch DISK proname 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 proname TEXT A3
COBOL proname (OSDECK APOST LIB
TXTLIB DEL utextlib proname
TXTLIB ADD utextlib proname
/* **********************************************************
** LINK PROGRAM MODULE                                    **
/* **********************************************************/
FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD loadlib LOADLIB A6 (RECFM V LRECL 1024 BLKSIZE 1024
FILEEXEC 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(proname)
- INCLUDE objlibl(IDMS)
- ENTRY proname
- NAME proname(R)
```

### Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>coblibvs</td>
<td><code>TEXTLIB p</code> Filename, filetype, and filemode of the library that contains host language compiler modules</td>
</tr>
<tr>
<td>create accmod</td>
<td>Filename of the file containing the CREATE ACCESS MODULE statement</td>
</tr>
<tr>
<td>linkctl</td>
<td>Filename of the file that contains the linkage editor control statements</td>
</tr>
<tr>
<td>loadlib</td>
<td>DDname of the load library containing the CA IDMS executable modules</td>
</tr>
</tbody>
</table>
### Variable Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>objlib</td>
<td>DDname of the user object library</td>
</tr>
<tr>
<td>objlib1</td>
<td>DDname of the CA IDMS object library</td>
</tr>
<tr>
<td>program</td>
<td>Filename, filetype, and filemode of the precompiler output</td>
</tr>
<tr>
<td>programe</td>
<td>Name of the user program</td>
</tr>
<tr>
<td>program source</td>
<td>Filename of the file containing the program source</td>
</tr>
<tr>
<td>sysidms1</td>
<td>DDname for the file of SYSIDMS parameters for the precompiler step</td>
</tr>
<tr>
<td>sysidms1 parms a</td>
<td>Filename of the file containing SYSIDMS parameters for the precompiler step</td>
</tr>
<tr>
<td>sysidms2</td>
<td>DDname for the file of SYSIDMS parameters for the step to create the access module</td>
</tr>
<tr>
<td>sysidms2 parms a</td>
<td>Filename of the file containing SYSIDMS parameters for the step to create the access module</td>
</tr>
<tr>
<td>sysipt1</td>
<td>DDname for the program source file</td>
</tr>
<tr>
<td>sysipt2</td>
<td>DDname for the file containing the CREATE ACCESS MODULE statement</td>
</tr>
<tr>
<td>syspch</td>
<td>DDname for the precompiler output</td>
</tr>
<tr>
<td>uloadlib</td>
<td>Filename, filetype, and filemode of the user load library</td>
</tr>
<tr>
<td>utextlib</td>
<td>Filename, filetype, and filemode of the user text library</td>
</tr>
</tbody>
</table>

### 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 IDMSMCL 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 Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dcmsg</code></td>
<td>DDname of the system message area (DDLDCMSG)</td>
</tr>
<tr>
<td><code>dictb</code></td>
<td>DDname of the application dictionary definition area (DDLDML)</td>
</tr>
<tr>
<td><code>dloddb</code></td>
<td>DDname of the application dictionary definition load area (DDLDCLOD)</td>
</tr>
<tr>
<td><code>idms.appldict.ddldclod</code></td>
<td>File name of the application dictionary definition load area (DDLDCLOD)</td>
</tr>
<tr>
<td><code>idms.appldict.ddldml</code></td>
<td>File name of the application dictionary definition area (DDLDML)</td>
</tr>
<tr>
<td><code>idms.sysmsg.ddldcmsg</code></td>
<td>File name of the system message area (DDLDCMSG)</td>
</tr>
<tr>
<td><code>idms.syssql.ddlcat</code></td>
<td>File name of the SQL definition area (DDLCAT) of the application dictionary</td>
</tr>
<tr>
<td><code>idms.syssql.ddlcatl</code></td>
<td>File name of the SQL definition load area (DDLCATLOD) of the application dictionary</td>
</tr>
<tr>
<td><code>idms.syssql.ddlcatx</code></td>
<td>File name of the SQL definition index area (DDLCATX) of the application dictionary</td>
</tr>
<tr>
<td><code>idms.tapejrnl</code></td>
<td>File name of the tape journal file</td>
</tr>
<tr>
<td><code>sqldd</code></td>
<td>DDname of the SQL definition area (DDLCAT) of the application dictionary</td>
</tr>
<tr>
<td><code>sqllod</code></td>
<td>DDname of the SQL definition load area (DDLCATLOD) of the application dictionary</td>
</tr>
<tr>
<td><code>sqlxdd</code></td>
<td>DDname of the SQL definition index area (DDLCATX) of the application dictionary</td>
</tr>
<tr>
<td><code>sysjrnl</code></td>
<td>DDname of the tape journal file</td>
</tr>
</tbody>
</table>

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. Type `XEDIT sysipt data a (NOPROF)` on the z/VM command line and press Enter
2. Type `INPUT` on the XEDIT command line and press Enter
3. Type in the IDMS/Pass input parameters in input mode
4. Press Enter to exit input mode
5. Type `FILE` on the XEDIT command line 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. Type `XEDIT sysidms data a (NOPROF)` on the z/VM command line and press Enter
2. Type `INPUT` on the XEDIT command line and press Enter
3. Type in the SYSIDMS parameters in input mode
4. Press Enter to exit input mode
5. Type `FILE` on the XEDIT command line and press Enter

⚠️ **Note:** For more information on documentation of SYSIDMS parameters, see the SYSIDMS Parameter File (https://docops.ca.com/display/IDMS19/SYSIDMS+Parameter+File).

---

**Test Database**

Complete information on the data in the test database, supplied with CA IDMS, to which most of the sample programs in this section refer, is presented in this section. You can use this information to develop SQL programs that access the test database.

- Table Names and Descriptions (see page 179)
- Test Data (see page 184)
- Test Database DDL (see page 188)
- Demo Data (see page 193)

---

**Table Names and Descriptions**

This section contains information for the following tables:

- ASSIGNMENT (see page 180)
- BENEFITS (see page 180)
- CONSULTANT (see page 181)
- COVERAGE (see page 181)
- DEPARTMENT (see page 181)
- DIVISION (see page 182)
- EMPLOYEE (see page 182)
- EXPERTISE (see page 182)
- INSURANCE_PLAN (see page 183)
ASSIGNMENT

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>Employee ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJ_ID</td>
<td>ID of project to which employee is assigned</td>
</tr>
<tr>
<td>START_DATE</td>
<td>Date employee was assigned to the project</td>
</tr>
<tr>
<td>END_DATE</td>
<td>Date employee completed work on the project</td>
</tr>
</tbody>
</table>

BENEFITS

| FISCAL_YEAR | Fiscal year for which this data applies |
| EMP_ID | Employee ID |
| VAC_ACCRUED | Vacation hours accrued to date |
| VAC_TAKEN | Vacation hours taken to date |
| SICK_ACCRUED | Sick days accrued to date |
| SICK_TAKEN | Sick days 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 | Hours of compensation time accrued |
| COMP_TAKEN | Hours 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 per pay period |
# Consultant Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON_ID</td>
<td>Unique consultant ID</td>
</tr>
<tr>
<td>CON_FNAME</td>
<td>Consultant's first name</td>
</tr>
<tr>
<td>CON_LNAME</td>
<td>Consultant's last name</td>
</tr>
<tr>
<td>MANAGER_ID</td>
<td>Employee ID of consultant's manager</td>
</tr>
<tr>
<td>DEPT_ID</td>
<td>ID of department to which consultant is assigned</td>
</tr>
<tr>
<td>PROJ_ID</td>
<td>ID of project to which consultant is assigned</td>
</tr>
<tr>
<td>STREET</td>
<td>Consultant's street address</td>
</tr>
<tr>
<td>CITY</td>
<td>Consultant's city</td>
</tr>
<tr>
<td>STATE</td>
<td>Consultant's state</td>
</tr>
<tr>
<td>ZIP_CODE</td>
<td>Consultant's zip code</td>
</tr>
<tr>
<td>PHONE</td>
<td>Consultant's phone</td>
</tr>
<tr>
<td>BIRTH_DATE</td>
<td>Birth date</td>
</tr>
<tr>
<td>START_DATE</td>
<td>Consultant's date of hire</td>
</tr>
<tr>
<td>SS_NUMBER</td>
<td>Social security number</td>
</tr>
<tr>
<td>RATE</td>
<td>Hourly rate of pay</td>
</tr>
</tbody>
</table>

# Coverage Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAN_CODE</td>
<td>Code of insurance plan providing the coverage</td>
</tr>
<tr>
<td>EMP_ID</td>
<td>Employee ID of employee having the coverage</td>
</tr>
<tr>
<td>SELECTION_DATE</td>
<td>Date employee selected this insurance plan</td>
</tr>
<tr>
<td>TERMINATION_DATE</td>
<td>Date employee terminated this insurance plan; if null, plan is still in force</td>
</tr>
<tr>
<td>NUM_DEPENDENTS</td>
<td>Number of dependents covered under this insurance plan</td>
</tr>
</tbody>
</table>

# Department Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT_ID</td>
<td>Unique department ID</td>
</tr>
<tr>
<td>DEPT_HEAD_ID</td>
<td>Employee ID of department head</td>
</tr>
<tr>
<td>DIV_CODE</td>
<td>Code of the division to which this department belongs</td>
</tr>
<tr>
<td>DEPT_NAME</td>
<td>Department name</td>
</tr>
</tbody>
</table>
## DIVISION

<table>
<thead>
<tr>
<th>DIV_CODE</th>
<th>Unique division ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV_HEAD_ID</td>
<td>Employee ID of division head</td>
</tr>
<tr>
<td>DIV_NAME</td>
<td>Division name</td>
</tr>
</tbody>
</table>

## EMPLOYEE

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>Unique employee ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANAGER_ID</td>
<td>Employee ID of employee's manager</td>
</tr>
<tr>
<td>EMP_FNAME</td>
<td>Employee's first name</td>
</tr>
<tr>
<td>EMP_LNAME</td>
<td>Employee's last name</td>
</tr>
<tr>
<td>DEPT_ID</td>
<td>ID of department to which employee is assigned</td>
</tr>
<tr>
<td>STREET</td>
<td>Employee's street address</td>
</tr>
<tr>
<td>CITY</td>
<td>Employee's city</td>
</tr>
<tr>
<td>STATE</td>
<td>Employee's state</td>
</tr>
<tr>
<td>ZIP_CODE</td>
<td>Employee's zip code</td>
</tr>
<tr>
<td>PHONE</td>
<td>Employee's phone</td>
</tr>
<tr>
<td>STATUS</td>
<td>Status of employee: (A) Active; (S) Short-term disability; (L) Long term disability</td>
</tr>
<tr>
<td>SS_NUMBER</td>
<td>Social security number</td>
</tr>
<tr>
<td>START_DATE</td>
<td>Employee's date of hire</td>
</tr>
<tr>
<td>TERMINATION_DATE</td>
<td>Date of termination</td>
</tr>
<tr>
<td>BIRTH_DATE</td>
<td>Birth date</td>
</tr>
</tbody>
</table>

## EXPERTISE

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>Employee ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKILL_ID</td>
<td>Skill ID</td>
</tr>
<tr>
<td>SKILL_LEVEL</td>
<td>Level of ability in this skill: 01 (low) to 04 (high)</td>
</tr>
<tr>
<td>EXP_DATE</td>
<td>Date this level of ability was achieved</td>
</tr>
</tbody>
</table>
## INSURANCE_PLAN

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAN_CODE</td>
<td>Unique plan code for company offering the insurance</td>
</tr>
<tr>
<td>COMP_NAME</td>
<td>Name of insurance company</td>
</tr>
<tr>
<td>STREET</td>
<td>Street address of insurance company</td>
</tr>
<tr>
<td>CITY</td>
<td>City address of insurance company</td>
</tr>
<tr>
<td>STATE</td>
<td>State address of insurance company</td>
</tr>
<tr>
<td>ZIP_CODE</td>
<td>Zip code of insurance company</td>
</tr>
<tr>
<td>PHONE</td>
<td>Telephone number of insurance company</td>
</tr>
<tr>
<td>GROUP_NUMBER</td>
<td>Commonwealth's group number for this insurance company</td>
</tr>
<tr>
<td>DEDUCT</td>
<td>Dollar amount deductible <strong>per year</strong> for this insurance plan</td>
</tr>
<tr>
<td>MAX_LIFE_BENEFIT</td>
<td>Maximum dollar amount to be paid to insured employee</td>
</tr>
<tr>
<td>FAMILY_COST</td>
<td>Amount deducted <strong>per paycheck</strong> for family coverage</td>
</tr>
<tr>
<td>DEP_COST</td>
<td>Additional amount deducted <strong>per paycheck</strong> per dependent</td>
</tr>
<tr>
<td>EFF_DATE</td>
<td>Date this coverage plan became effective</td>
</tr>
</tbody>
</table>

## JOB

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_ID</td>
<td>Unique job ID</td>
</tr>
<tr>
<td>JOB_TITLE</td>
<td>Job title</td>
</tr>
<tr>
<td>MIN_RATE</td>
<td>Minimum salary/hourly rate for this job</td>
</tr>
<tr>
<td>MAX_RATE</td>
<td>Maximum salary/hourly rate for this job</td>
</tr>
<tr>
<td>SALARY_IND</td>
<td>Indicator for type of salary: (S) salaried; (H) hourly</td>
</tr>
<tr>
<td>NUM_OF_POSITIONS</td>
<td>Total number of positions for this job</td>
</tr>
<tr>
<td>NUM_OPEN</td>
<td>Number of positions currently open</td>
</tr>
<tr>
<td>EFF_DATE</td>
<td>Date this job became effective</td>
</tr>
<tr>
<td>JOB_DESLINE_1</td>
<td>First line of job description</td>
</tr>
<tr>
<td>JOB_DESLINE_2</td>
<td>Second line of job description</td>
</tr>
</tbody>
</table>
**POSITION**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP_ID</td>
<td>Employee ID</td>
</tr>
<tr>
<td>JOB_ID</td>
<td>Job ID associated with this employee</td>
</tr>
<tr>
<td>START_DATE</td>
<td>Date employee began this job</td>
</tr>
<tr>
<td>FINISH_DATE</td>
<td>Date employee ended this job (null if current)</td>
</tr>
<tr>
<td>HOURLY_RATE</td>
<td>Hourly rate earned while in this job (if hourly position)</td>
</tr>
<tr>
<td>SALARY_AMOUNT</td>
<td>Yearly salary earned while in this job (if salaried position)</td>
</tr>
<tr>
<td>BONUS_PERCENT</td>
<td>Bonus percent amount for this position (if sales position)</td>
</tr>
<tr>
<td>COMM_PERCENT</td>
<td>Commission percent for this position (if sales position)</td>
</tr>
<tr>
<td>OVERTIME_RATE</td>
<td>Overtime rate for this position (if hourly position)</td>
</tr>
</tbody>
</table>

**PROJECT**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJ_ID</td>
<td>Unique project ID</td>
</tr>
<tr>
<td>PROJ_LEADER_ID</td>
<td>Employee ID of project leader</td>
</tr>
<tr>
<td>EST_START_DATE</td>
<td>Estimated date project is to begin</td>
</tr>
<tr>
<td>EST_END_DATE</td>
<td>Estimated date project is to end</td>
</tr>
<tr>
<td>ACT_START_DATE</td>
<td>Actual date project began</td>
</tr>
<tr>
<td>ACT_END_DATE</td>
<td>Actual date project ended</td>
</tr>
<tr>
<td>EST_MAN_HOURS</td>
<td>Total number of hours estimated for project</td>
</tr>
<tr>
<td>ACT_MAN_HOURS</td>
<td>Actual number of hours required for project</td>
</tr>
<tr>
<td>PROJ_DESC</td>
<td>Project description</td>
</tr>
</tbody>
</table>

**SKILL**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKILL_ID</td>
<td>Unique skill ID</td>
</tr>
<tr>
<td>SKILL_NAME</td>
<td>Skill name</td>
</tr>
<tr>
<td>SKILL_DESC</td>
<td>Skill description</td>
</tr>
</tbody>
</table>

**Test Data**

This section lists the test data stored in the test database for the following:

- Departments (see page 185)
**Departments**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Division code</th>
<th>Head ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3510</td>
<td>Appraisal - Used cars</td>
<td>D02</td>
<td>3082</td>
</tr>
<tr>
<td>2200</td>
<td>Sales - Used cars</td>
<td>D02</td>
<td>2180</td>
</tr>
<tr>
<td>1100</td>
<td>Purchasing - Used cars</td>
<td>D02</td>
<td>2246</td>
</tr>
<tr>
<td>3520</td>
<td>Appraisal - New cars</td>
<td>D04</td>
<td>3769</td>
</tr>
<tr>
<td>2210</td>
<td>Sales - New cars</td>
<td>D04</td>
<td>2010</td>
</tr>
<tr>
<td>4200</td>
<td>Leasing - New cars</td>
<td>D04</td>
<td>1003</td>
</tr>
<tr>
<td>1110</td>
<td>Purchasing - New cars</td>
<td>D04</td>
<td>1765</td>
</tr>
<tr>
<td>1120</td>
<td>Purchasing - Service</td>
<td>D06</td>
<td>2004</td>
</tr>
<tr>
<td>4600</td>
<td>Maintenance</td>
<td>D06</td>
<td>2096</td>
</tr>
<tr>
<td>3530</td>
<td>Appraisal - Service</td>
<td>D06</td>
<td>2209</td>
</tr>
<tr>
<td>5100</td>
<td>Billing</td>
<td>D06</td>
<td>2598</td>
</tr>
<tr>
<td>6200</td>
<td>Corporate Administration</td>
<td>D09</td>
<td>2461</td>
</tr>
<tr>
<td>5200</td>
<td>Corporate Marketing</td>
<td>D09</td>
<td>2894</td>
</tr>
<tr>
<td>5000</td>
<td>Corporate Accounting</td>
<td>D09</td>
<td>2466</td>
</tr>
<tr>
<td>4900</td>
<td>MIS</td>
<td>D09</td>
<td>2466</td>
</tr>
<tr>
<td>6000</td>
<td>Legal</td>
<td>D09</td>
<td>1003</td>
</tr>
<tr>
<td>4500</td>
<td>Human Resources</td>
<td>D09</td>
<td>3222</td>
</tr>
</tbody>
</table>

**Divisions**

<table>
<thead>
<tr>
<th>Division code</th>
<th>Division name</th>
<th>Head ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>D02</td>
<td>Used cars</td>
<td>2180</td>
</tr>
<tr>
<td>D04</td>
<td>New cars</td>
<td>2010</td>
</tr>
<tr>
<td>D06</td>
<td>Service</td>
<td>4321</td>
</tr>
<tr>
<td>D09</td>
<td>Corporate</td>
<td>1003</td>
</tr>
</tbody>
</table>
## Insurance Plans

<table>
<thead>
<tr>
<th>Plan ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLI</td>
<td>Providential Life Insurance</td>
</tr>
<tr>
<td>HHM</td>
<td>Homeostasis Health Maintenance Program</td>
</tr>
<tr>
<td>HGH</td>
<td>Holistic Group Health Association</td>
</tr>
<tr>
<td>DAS</td>
<td>Dental Associates</td>
</tr>
</tbody>
</table>

## Jobs

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Name</th>
<th>Minimum salary</th>
<th>Maximum salary</th>
<th>Salaried/ hourly</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8001</td>
<td>Vice president</td>
<td>90000</td>
<td>136000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4023</td>
<td>Accountant</td>
<td>44000</td>
<td>120000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>2051</td>
<td>AP Clerk</td>
<td>8.80</td>
<td>14.60</td>
<td>H</td>
<td>2</td>
</tr>
<tr>
<td>2053</td>
<td>AR Clerk</td>
<td>8.80</td>
<td>14.60</td>
<td>H</td>
<td>3</td>
</tr>
<tr>
<td>2077</td>
<td>Purch Clerk</td>
<td>17000</td>
<td>30000</td>
<td>S</td>
<td>3</td>
</tr>
<tr>
<td>3029</td>
<td>Computer Operator</td>
<td>25500</td>
<td>44000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>3051</td>
<td>Data Entry Clerk</td>
<td>8.50</td>
<td>11.45</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>6011</td>
<td>Manager - Acctng</td>
<td>59400</td>
<td>121000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4560</td>
<td>Mechanic</td>
<td>11.45</td>
<td>21.00</td>
<td>H</td>
<td>7</td>
</tr>
<tr>
<td>4666</td>
<td>Sr Mechanic</td>
<td>41000</td>
<td>91000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4734</td>
<td>Mkting Admin</td>
<td>25000</td>
<td>62000</td>
<td>S</td>
<td>2</td>
</tr>
<tr>
<td>3333</td>
<td>Sales Trainee</td>
<td>21600</td>
<td>39000</td>
<td>S</td>
<td>4</td>
</tr>
<tr>
<td>5555</td>
<td>Salesperson</td>
<td>30000</td>
<td>79500</td>
<td>S</td>
<td>9</td>
</tr>
<tr>
<td>6004</td>
<td>Manager - HR</td>
<td>66000</td>
<td>138000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>6021</td>
<td>Manager - Mktng</td>
<td>76000</td>
<td>150000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>2055</td>
<td>PAYROLL CLERK</td>
<td>17000</td>
<td>30000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4025</td>
<td>Writer - Mktng</td>
<td>31000</td>
<td>50000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>9001</td>
<td>President</td>
<td>111000</td>
<td>190000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4123</td>
<td>Recruiter</td>
<td>35000</td>
<td>56000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4130</td>
<td>Benefits Analyst</td>
<td>35000</td>
<td>56000</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>4012</td>
<td>Admin Asst</td>
<td>21000</td>
<td>44000</td>
<td>S</td>
<td>4</td>
</tr>
<tr>
<td>5111</td>
<td>CUST SER REP</td>
<td>27000</td>
<td>54000</td>
<td>S</td>
<td>4</td>
</tr>
<tr>
<td>4700</td>
<td>Purch Agent</td>
<td>33000</td>
<td>60000</td>
<td>S</td>
<td>5</td>
</tr>
<tr>
<td>5890</td>
<td>Appraisal Spec</td>
<td>45000</td>
<td>70000</td>
<td>S</td>
<td>5</td>
</tr>
</tbody>
</table>
### Job Information

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Name</th>
<th>Minimum salary</th>
<th>Maximum salary</th>
<th>Salaried/ hourly</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5110</td>
<td>CUST SERVICE MGR</td>
<td>40000</td>
<td>108000</td>
<td>S</td>
<td>1</td>
</tr>
</tbody>
</table>

### Projects

<table>
<thead>
<tr>
<th>Project ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P634</td>
<td>TV ads - WTVK</td>
</tr>
<tr>
<td>C200</td>
<td>New brand research</td>
</tr>
<tr>
<td>P400</td>
<td>Christmas media</td>
</tr>
<tr>
<td>C203</td>
<td>Consumer study</td>
</tr>
<tr>
<td>C240</td>
<td>Service study</td>
</tr>
<tr>
<td>D880</td>
<td>System analysis</td>
</tr>
</tbody>
</table>

### Skills

<table>
<thead>
<tr>
<th>Skill ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4444</td>
<td>Assembly</td>
</tr>
<tr>
<td>3333</td>
<td>Bodywork</td>
</tr>
<tr>
<td>3088</td>
<td>Brake work</td>
</tr>
<tr>
<td>3065</td>
<td>Electronics</td>
</tr>
<tr>
<td>1030</td>
<td>Acct Mgt</td>
</tr>
<tr>
<td>5130</td>
<td>Basic math</td>
</tr>
<tr>
<td>5160</td>
<td>Calculus</td>
</tr>
<tr>
<td>4250</td>
<td>Data entry</td>
</tr>
<tr>
<td>4370</td>
<td>Filing</td>
</tr>
<tr>
<td>5200</td>
<td>General Acctng</td>
</tr>
<tr>
<td>5500</td>
<td>General Mktng</td>
</tr>
<tr>
<td>5430</td>
<td>Mktng Writing</td>
</tr>
<tr>
<td>5420</td>
<td>Writing</td>
</tr>
<tr>
<td>4490</td>
<td>Gen Ledger</td>
</tr>
<tr>
<td>4430</td>
<td>Interviewing</td>
</tr>
<tr>
<td>1000</td>
<td>Management</td>
</tr>
<tr>
<td>4420</td>
<td>Telephone</td>
</tr>
<tr>
<td>5180</td>
<td>Statistics</td>
</tr>
<tr>
<td>4410</td>
<td>Typing</td>
</tr>
<tr>
<td>5309</td>
<td>Appraising</td>
</tr>
<tr>
<td>6770</td>
<td>Purchasing</td>
</tr>
</tbody>
</table>
Test Database DDL

This section 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,
CREATE TABLE DEPARTMENT
(DIV_CODE CHAR(03) NOT NULL,
DIV_NAME CHAR(40) NOT NULL)
IN SQLDEMO.INFOARA;

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),
GROUP NUMBER UNSIGNED NUMERIC(4,0) NOT NULL,
DEDUCT UNSIGNED DECIMAL(9,2),
MAX LIFEBENEFIT UNSIGNED DECIMAL(9,2),
FAMILY COST UNSIGNED DECIMAL(9,2),
DEP COST UNSIGNED DECIMAL(9,2),
EFF_DATE DATE NOT NULL)
IN SQLDEMO.INFOAREA;

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),
CREATE SCHEMA DEMOPROJ;
SET SESSION CURRENT SCHEMA DEMOPROJ;

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_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 DEMOEML.DEPARTMENT(DEPT_ID);
CREATE UNIQUE CALC KEY ON DEMOEML.DIVISION(DIV_CODE);
CREATE UNIQUE CALC KEY ON DEMOEML.EMPLOYEE(EMP_ID);
CREATE UNIQUE CALC KEY ON DEMOEML.INSURANCE_PLAN(PLAN_CODE);
CREATE UNIQUE CALC KEY ON DEMOEML.JOB(JOB_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJCONSULTANT(CON_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJ.PROJECT(PROJ_ID);
CREATE UNIQUE CALC KEY ON DEMOPROJSKILL(SKILL_ID);

*********************************************************************
* Create unique indexes for tables in order in which they were defined
*********************************************************************
CREATE UNIQUE INDEX AS_EMPROJ_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 DE_CODE_NDX ON DEMOEML.DEPARTMENT(DIV_CODE);
CREATE INDEX DI_HEAD_NDX ON DEMOEML.DIVISION(DIV_HEAD_ID);
CREATE INDEX EM_HEAD_NDX ON DEMOEML.DEPARTMENT(DEPT_HEAD_ID);
CREATE INDEX EM_MANAGER_NDX ON DEMOEML.EMPLOYEE(MANAGER_ID)
    IN SQLDEMO.INDXAREA;
CREATE INDEX EM_NAME_NDX ON DEMOEML.EMPLOYEE(EMP_LNAME,EMP_FNAME)
    IN SQLDEMO.INDXAREA;
CREATE INDEX EM_DEPT_NDX ON DEMOEML.EMPLOYEE(DEPT_ID)
    IN SQLDEMO.INDXAREA;
CREATE INDEX IN_NAME_NDX ON DEMOEML.INSURANCE_PLAN(COMP_NAME)
    COMPRESSED;
CREATE INDEX PO_JOB_NDX ON DEMOEML.POSITION(JOB_ID)
    IN SQLDEMO.INDXAREA;
CREATE INDEX CN_NAME_NDX ON DEMOPROJCONSULTANT(CON_LNAME,CON_FNAME);

*********************************************************************
* Create referential constraints
*********************************************************************
CREATE CONSTRAINT EMP_BENEFITS
    DEMOEML.BENEFITS (EMP_ID) REFERENCES
    DEMOEML.EMPLOYEE (EMP_ID)
    LINKED CLUSTERED
    ORDER BY (FISCAL_YEAR DESC);
CREATE CONSTRAINT INSPLAN_COVERAGE
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;
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 DEMOEMPL.PROJ.Assignment
DROP DEFAULT INDEX;
ALTER TABLE DEMOEMPL.PROJ.Consultant
DROP DEFAULT INDEX;
ALTER TABLE DEMOEMPL.PROJ.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
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;

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');
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 ('P534', 3411, '2000-02-01', '2000-03-01', null, null, 320, null, 'TV ads - WTVK');

INSERT INTO DEMOPROJ.PROJECT

INSERT INTO DEMOPROJ.PROJECT
VALUES ('P400', null, '2000-09-01', '2000-12-10', null, null, 2960, null, 'Christmas media');

INSERT INTO DEMOPROJ.PROJECT

INSERT INTO DEMOPROJ.PROJECT

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 (3029, 'Computer Operator', 25000, 44000, 'S', 1, '1989-03-01',
'Responsible for soliciting quotes from vendors', null);
'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 marketing plans', null);
INSERT INTO DEMOEMPL.JOB
values (9001, 'President', 111000, 190000, 'S', 1, '1984-01-01', 'Overall responsibility for well-being of 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', null);
'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
'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', 'Experience with general ledger');
INSERT INTO DEMOPROJ.SKILL
values (4490, 'Gen Ledger', 'Experience with general ledger');
INSERT INTO DEMOPROJ.SKILL
values (4420, 'Telephone', 'Basic customer support');
INSERT INTO DEMOPROJ.SKILL
values (5180, '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 DEMO_PROJ.SKILL
values (6666, 'Billing', 'Basic billing procedures');
INSERT INTO DEMO_PROJ.SKILL
values (6650, 'Diesel Engine Repair', 'Experience in diesel engine repair');
INSERT INTO DEMO_PROJ.SKILL
values (6670, 'Gas Engine Repair', 'Experience in gasoline engine repair');
INSERT INTO DEMO_PROJ.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
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
values (1003, null, 'Patricia', 'Bennett', 5000, '152B Central St', 'Medford', 'MA', '02432', '5088487709', 'A', 098339556, '1991-10-29', null, '1963-12-23');
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
INSERT INTO DEMOEMPL.EMPLOYEE
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', 5000,
'4 Birch St', 'Stoneham', 'MA', '02113',
null, 'A', 059975848, '1998-12-07',
null, '1968-11-29');

INSERT INTO DEMOEMPL.EMPLOYEE
values (2246, 2466, 'Marylou', 'Hamel', 1100,
'11 Main St', 'Medford', 'MA', '02066',
'6176642209', 'A', 033765632, '1997-11-12',
null, '1971-12-28');

INSERT INTO DEMOEMPL.EMPLOYEE
values (4703, 2246, 'Martin', 'Halloran', 1100,
'27 Elm St', 'Brookline', 'MA', '02066',
'6176648290', 'A', 054475888, '1997-03-19',
null, '1973-06-07');

INSERT INTO DEMOEMPL.EMPLOYEE
values (3802, 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, 3802, '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', '02066',
'5083145366', 'A', 066783225, '1989-10-28',
nul, '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',
nul, '1964-10-28');

INSERT INTO DEMOEMPL.EMPLOYEE
values (3767, 2180, 'Frank', 'Lowe', 2200,
'25 Rutland St', 'Natick', 'MA', '02364',
'5082844094', 'A', 066905009, '1994-08-31',
nul, '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',
nul, '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',
nul, '1961-11-23');

INSERT INTO DEMOEMPL.EMPLOYEE
CA IDMS - 19.0


INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE

INSERT INTO DEMOEMPL.EMPLOYEE
values (3294, 2004, 'Carolyn', 'Johnson', 1120,
<table>
<thead>
<tr>
<th>ID</th>
<th>Dept</th>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>ZIP</th>
<th>HireDate</th>
<th>Ter Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td></td>
<td>Mark</td>
<td>5612</td>
<td>617-556-3243</td>
<td>'560 Camden St', 'Canton', 'MA', '02020', 1993-06-17</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3338</td>
<td>2004</td>
<td>Mike</td>
<td>9876</td>
<td>508-555-4321</td>
<td>'301 Summer St', 'Brookline', 'MA', '02066', 1990-06-17</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Carl</td>
<td>1234</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Alice</td>
<td>5678</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Art</td>
<td>7890</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Bob</td>
<td>3456</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Carol</td>
<td>1011</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Dan</td>
<td>2020</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Susan</td>
<td>3030</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Fred</td>
<td>4040</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Lauren</td>
<td>5050</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Greg</td>
<td>6060</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Helen</td>
<td>7070</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2209</td>
<td>2894</td>
<td>Eric</td>
<td>8080</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3341</td>
<td>2209</td>
<td>Margaret</td>
<td>9090</td>
<td>617-923-8845</td>
<td>'18 South St', 'Newton', 'MA', '02456', 1993-07-02</td>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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, 5555, '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 (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, 53665.56, 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 (2461, 4012, '1991-09-09', null, null, 43784.00, null, null, null);
INSERT INTO DEMOEMPL.POSITION
values (3841, 4012, '1994-10-25', null, null, 33000.00, 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, '1994-02-01', 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 (3411, 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, .105, 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, 28.85, null);
INSERT INTO DEMOEMPL.POSITION
values (3767, 3333, '1994-08-31', '1995-01-10', null, 2200.00, null, null, null );
INSERT INTO DEMOEMPL.POSITION
values (2180, 3333, '1997-10-27', '1990-04-17', null, 10000.10, null, .09, null);
INSERT INTO DEMOEMPL.POSITION
values (4660, 3333, '1997-03-30', '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, '1993-10-01', '1993-10-01', 9.50, null, null, 13.50, null);
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 (3508, 6770, '04', '1998-01-31' );
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 (5900, 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 (2437, 3333, '04', '1997-03-15');
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 (2804, 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');
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 (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.CONSULTANT
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.CONSULTANT
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.CONSULTANT
values (9388, 'Linda', 'Candido', 2466, 5200, 'D880',
'S4 Church St', 'Newton', 'MA', '02456',
'6179943082', '1959-08-30', '1997-12-21', 033006132,
76.00 );
INSERT INTO DEMOProj.CONSULTANT
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,
'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 ,
'2000-10-15', .05, null,
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 4660, 68, 56, 8.00, 0, .06, null, 'STOCK', .05, 2100.00, 16, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
values (2000, 3767, 68, 68, 8.00, .07, 2250, '2000-09-22', .045, null, '401K', .05, 850.68, 0, 0, 'JRCOLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
values (2000, 2448, 68, 20.50, 8.00, 0, .075, 6600, '2000-07-13', .05, null, 'BONDS', .04, 2100.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
values (2000, 3704, 68, 80, 8.00, .08, 3060, '2000-10-30', .06, null, 'STOCK', .05, 1406.90, 32.5, 16.0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
values (2000, 4008, 68, 0, 8.00, .10, 2000, '2000-07-14', .04, null, '401K', .05, 1750.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
values (2000, 4962, 68, 16, 8.00, .06, 3060, '2000-07-14', .04, null, '401K', .05, 1406.90, 32.5, 16.0, 'COLL', null, 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, 1.00, .08, 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, 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, 1456.00, 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', 'HP302-7409', 50.50);
INSERT INTO DEMOEMPL.BENEFITS
values (2000, 1034, 92.5, 72, 8.00, 2.5, .10, 5300, '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,
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, 92.5, 32, 8.00, 5.5, 0, 0,
        '2000-06-14', .06, null,
        '401K', .04, 2100.00, 0.0,
        'COLL', 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, 3841, 68, 0, 8.00, 2.00, 0, 0,
        '2000-10-10', .06, null,
        NULL, NULL, 2100.00, 0.0,
        'HSDIP', 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, 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, 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, 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, 750.00, 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, '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, .04, 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, 10, 8, 0, 0, 0, 0, 0, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 4703, 80, 80, 10, 8, 7, 0, 0, 0, 0, 0, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1997, 4703, 80, 80, 10, 8, 6, 0, 0, 0, 0, 0, 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, NULL, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 5008, 48, 48, 8, 7, .10, 1500, 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', 0.04, NULL, '401K', 0.03, 1200.00, 0, 0, 'HSIDP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3769, 120, 120, 15, 8.5, 0, 0, '1998-09-01', 0.04, NULL, '401K', 0.04, 1100.00, 0, 0, 'HSIDP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4001, 120, 120, 15, 2, 0, 0, '1999-12-01', 0.045, NULL, NULL, NULL, 1500.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4008, 120, 120, 15, 2, 0, 0, '1999-01-15', 0.04, NULL, '401K', 0.05, 1500.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4008, 80, 80, 15, 1, 0, 0, '1998-01-31', 0.035, NULL, '401K', 0.05, 1350.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4008, 80, 72, 15, 0, 0, 0, '1997-01-30', 0.035, 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', 0.06, NULL, '401K', 0.05, 1150.50, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4962, 80, 80, 10, 1, 0, 0, '1998-10-16', 0.05, NULL, '401K', 0.05, 1000.00, 2, 2, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4962, 12, 0, 2, 0, 0.05, 3000, 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', 0.055, NULL, 'STOCK', 0.05, 2100.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2010, 160, 152.5, 15, 3, 0, 0, '1998-03-30', 0.05, NULL, 'STOCK', 0.05, 2000.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2010, 160, 160, 15, 3, 0, 0, '1997-03-10', 0.05, NULL, 'BONDS', 0.05, 1600.00, 2, 2, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3764, 120, 120, 15, 2, 0, 0, '1999-08-01', 0.05, '1991-05-10', 'STOCK', 0.06, 1500.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3764, 120, 120, 15, 3, 0, 0, '1998-08-30', 0.05, '1991-05-10', 'STOCK', 0.05, 1200.00, 14, 14, 'COLL', NULL, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3764, 120, 120, 15, 5, 0, 0, '1997-08-17', 0.045, '1991-05-10', 'STOCK', 0.05, 1000.00, 0, 0, 'COLL', null, null);
null, null);

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 5090, 80, 80, 15, 2, 0, '1999-07-30', .035, NULL, NULL, 800.00, 0, 0, 'JRCOLL', NULL, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 5090, 24, 24, 15, 0, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 0, 0, 'JRCOLL', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4027, 120, 120, 15, 8, 0, '1999-03-15', .03, NULL, NULL, 1500.00, 16, 16, 'COLL', NULL, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4027, 120, 120, 15, 0, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 0, 0, 'COLL', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4027, 80, 80, 10, 2.5, 0, '1997-04-01', .03, NULL, NULL, 1000.00, 0, 0, 'COLL', NULL, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3991, 120, 120, 15, 8, 0, '1999-12-04', .045, '1995-06-05', '401K', .05, 1300.00, 0, 0, 'COLL', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3991, 120, 116, 15, 2, 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, '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, '1999-11-15', .07, null, '401K', .08, 2500.50, 36, 0, 'COLL', 'HSDIP', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 1765, 160, 160, 15, 0, .10, '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, '1997-10-30', .065, null, '401K', .07, 2400.00, 72, 0, 'COLL', 'HSDIP', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2106, 160, 160, 15, .95, .07, '1999-05-01', .05, '1999-08-17', 'BONDS', .04, 1800.00, 0, 0, 'HSDIP', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2106, 160, 160, 15, 3, 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, '1997-04-30', .03, NULL, NULL, 1700.00, 0, 0, 'HSDIP', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2096, 160, 128, 15, 3, .04, '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, '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, '1997-02-15', .06, NULL, NULL, 1700.00, 0, 0, 'HSDIP', null, null);
```

```
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2437, 120, 16, 15, 11.5, 0, '1999-08-01', null, null);
```

```
```
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, 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, 0, 0, '1999-01-30',
.035, NULL, NULL, NULL, 2150.50, 0, 0, 'GED',
'MC655-6901', 54.86);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2598, 120, 120, 15, 15, 0, 0, '1998-01-15',
.03, NULL, NULL, NULL, 1800.00, 0, 0, 'GED',
'MC655-6901', 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, 'GED',
'MC655-6901', 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, 8.5, 0, 0, '1999-09-15',
.05, NULL, NULL, NULL, 2500.00, 0, 0, 'HSDIP',
'MC655-4490', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3778, 120, 120, 15, 10, 0, 0, '1998-09-01',
.05, NULL, NULL, NULL, 1000.00, 0, 0, 'HSDIP',
'MC655-4490', 79.62);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3778, 120, 120, 15, 10, 0, 0, '1997-09-18',
.05, NULL, NULL, NULL, 1000.00, 0, 0, 'HSDIP',
'MC655-4490', 45.75);
INSERT INTO DEMOEMPL.BENEFITS
.05, NULL, 'BONDS', .06, 2850.60, 0, 0, 'HSDIP',
'MC655-4490', 84.05);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 1034, 160, 112, 15, 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, 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, '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, 1550.00, 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, 1600.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1999, 4456, 80, 80, 15, 3, 0, 0, '1999-02-17', .03, NULL, NULL, 650.00, 0, 0, 'HSDIP', 'MC655-6680', .04, 1600.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 4456, 80, 80, 10, 0, 0, '1998-02-17', .02, NULL, NULL, 700.00, 0, 0, 'HSDIP', 'MC655-6680', .04, 1600.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1997, 4456, 48, 48, 8, 1, 0, 0, null, null, NULL, NULL, 1600.00, 0, 0, 'HSDIP', 'MC655-6680', .04, 1600.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1999, 3288, 120, 120, 15, 9, 0, 0, '1999-02-01', .035, NULL, NULL, 1380.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 3288, 120, 120, 15, 3, 0, 0, '1998-02-13', .035, NULL, NULL, 1250.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1997, 3288, 120, 120, 15, 6.5, 0, 0, '1997-02-04', .05, NULL, '401K', .06, 1350.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1999, 3341, 120, 120, 15, 9.0, 0, 0, '1999-07-25', .05, NULL, '401K', .06, 1350.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 3341, 120, 120, 15, 8.0, 0, 0, '1998-07-26', .06, NULL, '401K', .06, 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, 900.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1999, 2209, 120, 120, 15, 6.0, 0, 0, '1999-07-02', .05, NULL, '401K', .06, 1200.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 2209, 120, 120, 15, 5, 0, 0, '1998-06-17', .05, NULL, '401K', .06, 1200.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1997, 2209, 120, 120, 15, 5, 0, 0, '1997-06-28', .045, null, null, 1550.80, 8, 8, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', 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, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1998, 3294, 120, 120, 15, 7, 0, 0, '1998-06-01', .05, NULL, '401K', .03, 1380.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1997, 3294, 120, 120, 15, 3, 0, 0, '1997-06-28', .045, null, null, 1550.80, 8, 8, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', null, null);  
INSERT INTO DEMOEMPL.BENEFITS  
VALUES (1999, 3294, 120, 120, 15, 13, 0, 0, '1998-01-28', .05, NULL, '401K', .03, 1100.00, 0, 0, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', 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, 'HSDIP', 'MC655-4402', .04, 1250.00, 0, 0, 'JRCOLL', 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',
.05, NULL, '401K', .04, 1900.00, 0, 0, 'JRCOLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
.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',
null, NULL, 'BONDS', .08, 1350.60, 0, 0, 'COLL', null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3118, 120, 120, 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',
.06, 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',
.04, '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, '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, '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, '1999-10-05', 0.06, NULL, 'BONDS', 0.05, 1200.00, 0, 0, 'JRCOLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3841, 120, 120, 15, 3, 0, '1998-10-31', 0.05, NULL, 'BONDS', 0.05, 1020.00, 0, 0, 'JRCOLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3841, 80, 80, 15, 2, 0, '1997-10-11', 0.07, NULL, NULL, NULL, NULL, 980.00, 0, 0, 'JRCOLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4002, 120, 120, 15, 3, 0, '1999-12-01', 0.05, NULL, NULL, 1630.00, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 4002, 120, 120, 15, 6, 0, '1998-12-05', 0.04, NULL, NULL, 1400.00, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4002, 120, 120, 15, 5, 0, '1997-12-01', 0.04, NULL, NULL, 1380.00, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 1003, 160, 56, 2, 0, 0, '1999-12-01', 0.10, NULL, '401K', 0.10, NULL, 0, 0, 'MAS', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 1003, 160, 80, 15, 0, 0, '1998-10-18', 0.05, NULL, '401K', 0.03, 1800.00, 16, 16, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 1003, 160, 120, 15, 0, 0, '1997-12-18', 0.03, NULL, '401K', 0.03, 1000.00, 16, 16, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2781, 120, 120, 15, 8, 0, '1999-04-11', 0.05, NULL, '401K', 0.03, 1700.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2781, 120, 104, 15, 8, 0, '1998-12-02', 0.05, NULL, '401K', 0.03, 1100.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2781, 120, 112, 15, 9, 0, '1997-12-18', 0.03, NULL, NULL, 880.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3449, 120, 120, 15, 8, 0, '1999-12-08', 0.04, NULL, NULL, 240.50, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3449, 120, 104, 15, 8, 0, '1998-12-02', 0.05, NULL, '401K', 0.03, 1100.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3449, 120, 112, 15, 9, 0, '1997-12-18', 0.03, NULL, NULL, 880.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2466, 120, 120, 15, 3, 0, '1999-12-01', 0.05, NULL, '401K', 0.03, 1600.00, 0, 0, 'HSDIP', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2466, 120, 112, 15, 10, 0, '1998-10-18', 0.04, NULL, NULL, 1300.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2466, 120, 120, 15, 10, 0, '1997-10-10', 0.035, NULL, NULL, 980.00, 0, 0, 'JRCOLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 3449, 120, 120, 15, 8, 0, '1999-12-08', 0.04, NULL, NULL, 240.50, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 3449, 120, 104, 15, 8, 0, '1998-12-02', 0.05, NULL, '401K', 0.03, 1100.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 3449, 120, 112, 15, 9, 0, '1997-12-18', 0.03, NULL, NULL, 880.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 2781, 120, 120, 15, 8, 0, '1999-04-11', 0.05, NULL, '401K', 0.03, 1700.00, 0, 0, 'COLL', null, null);

INSERT INTO DEMOEMPL.BENEFITS
VALUES (1998, 2781, 120, 96, 15, 15, 0, '1998-04-26',
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2781, 120, 120, 15, 2.0, 0, '1997-04-18',
        .05, 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, 1200.00, 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, 1200.00, 22, 0, 'MAS',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 2894, 120, 15, 0, 0.0, '1997-05-11',
        .06, NULL, 'STOCK', .08, 1200.00, 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, 1.0, 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, 1100.00, 0.0, 'JRCOLL',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1999, 4358, 120, 112, 15, 2.0, 0.07, 1300, '1999-10-01',
        .055, 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, 820.00, 0.0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.BENEFITS
VALUES (1997, 4358, 80, 80, 15, 14.5, 0.06, 980, '1997-09-26',
        .055, 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, 1200.00, 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, 0.05, 7690, '1997-04-01',
        .06, NULL, 'BONDS', .06, 1580.50, 48, 0, 'HSDIP',
        null, null);
INSERT INTO DEMOEMPL.INSURANCE_PLAN
values ('PLI', 'Providential Life Insurance',
        '950 Gibraltar Ave', 'Lisbon', 'VA', '03097',
        '7033548300', 7815, null, 1000000, null, null, '1988-02-01');
INSERT INTO DEMOEMPL.INSURANCE_PLAN
values ('HHM', 'Homeostasis 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 ('HHM', 2598, '1997-07-25',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 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 ('HHM', 1034, '1993-12-01',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2424, '1993-07-24',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 4456, '1994-01-04',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3288, '1995-10-20',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3288, '1995-10-20',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3341, '1996-01-23',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3341, '1997-01-01',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2209, '1992-08-12',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 2209, '1993-12-01',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3294, '1993-02-19',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3338, '1994-12-11',
null, 0);
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 ('HHM', 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, '1995-10-20',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 4001, '1995-10-20',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 4008, '1996-01-23',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4008, '1996-01-23',
null, 1);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4962, '1997-10-04',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 4962, '1997-10-04',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('DAS', 3764, '1994-08-25',
null, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 5090, '1998-07-12',
null, 0);
null, 3);
INSERT INTO DEMOEMPL.COVERAGE values ('PLI', 4027, '1996-04-01', NULL, 0);
INSERT INTO DEMOEMPL.COVERAGE values ('HHM', 3991, '1994-11-12', '1995-12-31', 5);
INSERT INTO DEMOEMPL.COVERAGE values ('HGH', 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, 0);
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, 0);
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
SELECT 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;

Precompiler Directives

Information about CA IDMS precompiler directives that are not associated with SQL statements and
host variable declarations is presented in this section.

- Overriding DDLDML Area Ready Mode (see page 221)
- No Logging of Program Activity Statistics (see page 222)
- Generating a Source Listing (see page 222)

Overriding DDLDML Area Ready Mode

Syntax

```c
*RETRIEVAL
*PROTECTED-UPDATE
```

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 read 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.
No Logging of Program Activity Statistics

Syntax

```
*NO-ACTIVITY-LOG
```

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

Generating a Source Listing

Syntax

```
*DMLIST
*NODMLIST
```

Parameters

- **DMLIST**
  Specifies that the source listing is to be generated for the statements that follow. *DMLIST overrides a previous *NODMLIST directive and the NLIST precompiler parameter.
**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.

- **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.