Embedded SQL--an ugly example (do not write a program like this...for purposes of argument ONLY)

```
/* -To get right to it, here is an example program that uses Embedded
    SQL. Embedded SQL allows programmers to connect to a database and
    include SQL code right in the program, so that their programs can
    use, manipulate, and process data from a database.
   -This example C Program (using Embedded SQL) will print a report.
   -This program will have to be precompiled for the SQL statements,
    before regular compilation.
   -The EXEC SQL parts are the same (standard), but the surrounding C
    code will need to be changed, including the host variable
    declarations, if you are using a different language.
   -Embedded SQL changes from system to system, so, once again, check
    local documentation, especially variable declarations and logging
    in procedures, in which network, DBMS, and operating system
    considerations are crucial. */

/************************************************/
/* THIS PROGRAM IS NOT COMPILABLE OR EXECUTABLE */
/* IT IS FOR EXAMPLE PURPOSES ONLY              */
/************************************************/

#include <stdio.h>

/* This section declares the host variables; these will be the
   variables your program uses, but also the variable SQL will put
   values in or take values out. */
EXEC SQL BEGIN DECLARE SECTION;
  int BuyerID;
  char FirstName[100], LastName[100], Item[100];
EXEC SQL END DECLARE SECTION;

/* This includes the SQLCA variable, so that some error checking can be done.
*/
EXEC SQL INCLUDE SQLCA;

main() {

/* This is a possible way to log into the database */
EXEC SQL CONNECT UserID/Password;

/* This code either says that you are connected or checks if an error
   code was generated, meaning log in was incorrect or not possible. */
  if(sqlca.sqlcode) {
    printf(Printer, "Error connecting to database server.\n");
    exit();
  }
  printf("Connected to database server.\n");

/* This declares a "Cursor". This is used when a query returns more
   than one row, and an operation is to be performed on each row
   resulting from the query. With each row established by this query,
   I'm going to use it in the report. Later, "Fetch" will be used to
   pick off each row, one at a time, but for the query to actually
   be executed, the "Open" statement is used. The "Declare" just
   establishes the query. */
EXEC SQL DECLARE ItemCursor CURSOR FOR
  SELECT ITEM, BUYERID
  FROM ANTIQUES
  ORDER BY ITEM;
EXEC SQL OPEN ItemCursor;

/* +-- You may wish to put a similar error checking block here --+ */

/* Fetch puts the values of the "next" row of the query in the host
   variables, respectively. However, a "priming fetch" (programming
   technique) must first be done. When the cursor is out of data, a
   sqlcode will be generated allowing us to leave the loop. Notice
   that, for simplicity's sake, the loop will leave on any sqlcode,
   even if it is an error code. Otherwise, specific code checking must
   be performed. */
EXEC SQL FETCH ItemCursor INTO :Item, :BuyerID;
  while(!sqlca.sqlcode) {

/* With each row, we will also do a couple of things. First, bump the
   price up by $5 (dealer's fee) and get the buyer's name to put in
   the report. To do this, I'll use an Update and a Select, before
   printing the line on the screen. The update assumes however, that
   a given buyer has only bought one of any given item, or else the
   price will be increased too many times. Otherwise, a "RowID" logic
   would have to be used (see documentation). Also notice the colon
   before host variable names when used inside of SQL statements. */

EXEC SQL UPDATE ANTIQUES
  SET PRICE = PRICE + 5
  WHERE ITEM = :Item AND BUYERID = :BuyerID;

EXEC SQL SELECT OWNERFIRSTNAME, OWNERLASTNAME
  INTO :FirstName, :LastName
  FROM ANTIQUEOWNERS
  WHERE BUYERID = :BuyerID;

    printf("%25s %25s %25s", FirstName, LastName, Item);

/* Ugly report--for example purposes only! Get the next row. */
EXEC SQL FETCH ItemCursor INTO :Item, :BuyerID;
  }

/* Close the cursor, commit the changes (see below), and exit the
   program. */
EXEC SQL CLOSE ItemCursor;
EXEC SQL COMMIT RELEASE;
  exit();
}
```

vars

connect

declare cursor for multiple line reads

Read data

update tuples

read one row

close connection

ends the current transaction and makes changes to the database permanent

~~release~~ frees all resources and logs off the database & disconnect from Oracle

syntax

EXEC SQL CONNECT :username_password

; ~~user~~name IDENTIFIED BY : password

AT db_name USING :dbstring

Source:
http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.embed.doc/doc/c0007014.htm

Embedded SQL application template in C

This is a simple embedded SQL application that is provided for you to use to test your embedded SQL development environment and to help you learn about the basic structure of embedded SQL applications.

Embedded SQL applications require the following structure:

- including the required header files
- host variable declarations for values to be included in SQL statements
- a database connection
- the execution of SQL statements
- the handling of SQL errors and warnings related to SQL statement execution
- dropping the database connection

The following source code demostrates the basic structure required for embedded SQL applications written in C.

Figure 16. Sample program: template.sqc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>

EXEC SQL BEGIN DECLARE SECTION;
  short id;
  char name[10];
  short dept;
  double salary;
  char hostVarStmtDyn[50];
EXEC SQL END DECLARE SECTION;

int main()
{
  int rc = 0;
  EXEC SQL INCLUDE SQLCA;

  /* connect to the database */
  printf("\n Connecting to database...");
  EXEC SQL CONNECT TO "sample";
  if (SQLCODE < 0)
  {
     printf("\nConnect Error:  SQLCODE = %ld \n", SQLCODE);
     goto connect_reset;
  }
  else
  {
     printf("\n Connected to database.\n");
  }

  /* execute an SQL statement (a query) using static SQL; copy the single row
     of result values into host variables*/
  EXEC SQL SELECT id, name, dept, salary
           INTO :id, :name, :dept, :salary
           FROM staff WHERE id = 310;
  if (SQLCODE < 0)
  {
     printf("Select Error:  SQLCODE = %ld \n", SQLCODE);
  }
  else
  {
     /* print the host variable values to standard output */
     printf("\n Executing a static SQL query statement, searching for
        \n the id value equal to 310\n");
     printf("\n ID    Name          DEPT       Salary\n");
     printf(" %3d %7s %7d %16.2f\n\n", id, name, dept, salary);
  }

  strcpy(hostVarStmtDyn, "UPDATE staff
                          SET salary = salary + 1000
                          WHERE dept = ?");
  /* execute an SQL statement (an operation) using a host variable
     and DYNAMIC SQL*/
  EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
  if (SQLCODE < 0)
  {
     printf("Prepare Error:  SQLCODE = %ld \n", SQLCODE);
  }
  else
  {
     EXEC SQL EXECUTE StmtDyn USING :dept;
  }
  if (SQLCODE < 0)
  {
     printf("Execute Error:  SQLCODE = %ld \n", SQLCODE);
  }

  /* Read the updated row using STATIC SQL and CURSOR */
  EXEC SQL DECLARE posCur1 CURSOR FOR
     SELECT id, name, dept, salary
     FROM staff WHERE id = 310;
  if (SQLCODE < 0)
  {
     printf("Declare Error:  SQLCODE = %ld \n", SQLCODE);
  }
  EXEC SQL OPEN posCur1;
  EXEC SQL FETCH posCur1 INTO :id, :name, :dept, :salary ;
  if (SQLCODE < 0)
  {
     printf("Fetch Error:  SQLCODE = %ld \n", SQLCODE);
  }
  else
  {
     printf(" Executing an dynamic SQL statement, updating the
              \n salary value for the id equal to 310\n");
     printf("\n ID    Name          DEPT       Salary\n");
     printf(" %3d %7s %7d %16.2f\n\n", id, name, dept, salary);
  }

  EXEC SQL CLOSE posCur1;

  /* Commit the transaction */
```

```
   printf("\n  Commit the transaction.\n");
   EXEC SQL COMMIT;
   if (SQLCODE < 0)
   {
      printf("Error:  SQLCODE = %ld \n", SQLCODE);
   }

   /* Disconnect from the database */
   connect_reset :
      EXEC SQL CONNECT RESET;
      if (SQLCODE < 0)
      {
         printf("Connection Error:  SQLCODE = %ld \n", SQLCODE);
      }
   return 0;
} /* end main */
```

ends the current transaction and makes changes to the database permanent

Notes to Figure 16:

Note	Description
1	Include files: This directive includes a file into your source application.
2	Declaration section: Declaration of host variables that will be used to hold values referenced in the SQL statements of the C application.
3	Local variable declaration: This block declares the local variables to be used in the application. These are not host variables.
4	Including the SQLCA structure: The SQLCA structure is updated after the execution of each SQL statement. This template application uses certain SQLCA fields for error handling.
5	Connection to a database: The initial step in working with the database is to establish a connection to the database. Here, a connection is made by executing the CONNECT SQL statement.
6	Error handling: Checks to see if an error occurred.
7	Executing a query: The execution of this SQL statement assigns data returned from a table to host variables. The C code below the SQL statement execution prints the values in the host variables to standard output.
8	Executing an operation: The execution of this SQL statement updates a set of rows in a table identified by their department number. Preparation (performed three lines above) is a step in which host variable values, such as the one referenced in this statement, are bound to the SQL statement to be executed.
9	Executing an operation: In this line and the previous line, this application uses cursors in static SQL to select information in a table and print the data. After the cursor is declared and opened, the data is fetched, and finally the cursor is closed.

Note	Description
	Commit the transaction: The COMMIT statement finalizes the database changes that were made within a unit of work.
	And finally, the database connection must be dropped.

Related concepts

- Error message retrieval in embedded SQL applications
- Include files and definitions required for embedded SQL applications
- Executing SQL statements in embedded SQL applications

Related tasks

- Declaring host variables in embedded SQL applications
- Setting up the embedded SQL development environment

Related reference

- COMMIT statement
- ROLLBACK statement
- SQLCA (SQL communications area)