

Oracle Studio #2 -- Embedded SQL and Oracle

CSCI-4380 Database Systems

Fall 2001

1. Introduction

In this studio, you will continue to use the Oracle database system. In particular you will investigate the use of embedded SQL with Oracle. The name of the embedded SQL interface for Oracle is Pro*C. The programming language that you will use is the C programming language (it is also possible to use C++ with Pro*C). Don't worry if you do not know C well. You will be given most of the program and all that you need to do is add the required embedded SQL statements. Familiarity with Microsoft Visual C/C++ is a help, but again not a requirement.

This studio will also use *The Movies Database* (the full database). In fact, we will return to the game *Six Degrees of Kevin Bacon*. The program that you complete in this studio finds and displays the titles of all movies in which two given actors played roles. Such a program might be useful to you as you play the Kevin Bacon game and try to construct longer paths from Kevin Bacon to other actors. For example, when given the names Woody Allen and Diane Keaton, the program produces the following output:

```
Joint Films for Woody Allen and Diane Keaton
-----
Play it Again, Sam
Sleeper
Love and Death
Annie Hall
Manhattan
Manhattan Murder Mystery
```

2. Getting Started

The ".pc" file extension is used for Pro*C files, that is, files with embedded SQL in a C program. A file `Program.pc` contains a C program for computing the names of films in which two given actors played roles. The only thing missing from the file is the required embedded SQL. In place of each SQL statement is a block of comments describing what the embedded SQL should do. Your job is to replace these comment blocks with the required embedded SQL statements. You must then get the resulting program to run.

The C compiler that we will use with Pro*C for this studio is Microsoft Visual C/C++. Everything has been set up for you in a workspace that you must copy to your computer. To do this, use a web browser to download the file

<http://www.cs.rpi.edu/~sibel/dbs/studios/5/Studio5.zip>

to your computer. When you download this file, be sure that it is placed on the C: drive as file `C:\Studio5.Zip`. Placing it anywhere else will lead to error messages from the compiler when you try to compile it later.

The file is a zip file, which is a compressed file containing a folder with many things in it. You must unzip (decompress) the file before you can use it. To do this, double click on the file icon for the file. Depending on what tool is installed on your computer for unzipping files, either the file will automatically unzip itself or the zip utility window will appear. If the zip utility window appears, you will see near the top of it a button labeled Extract. Click on this Extract button. In the resulting window, click on the line for the C: drive. When you do this a button appears in the lower right corner of the window labeled New Folder. Click on this New Folder button and type Studio5 in the text entry box that appears. Then click on the OK button. This says that you want to uncompress the zip file in a new folder named Studio5 on the C: disk. Finally, click the Extract button in the upper right corner of the window to uncompress the files into this new folder. As mentioned above, some unzip tools will do all this work automatically when you double click on the icon for the Studio5.Zip file.

Now look for the Microsoft Visual C++6 icon on the Desktop and double click on it to start Microsoft Visual C++. Once it starts, click on the File menu at the top of the window and then on the Open Workspace entry in the menu that pops down. In the dialogue box that appears search for the file ESQlProj.dsw in the C:\Studio5 folder that you just created. This is the Microsoft Visual C++ workspace that contains everything that you need for this studio. Open this workspace file.

3. Configuring Visual C++

Before going any further, it is necessary to configure Visual C++ so that it can find the Pro*C precompiler and the required include and link libraries. To do this, first look in the file system on your computer to determine where the orant directory is. If you are using one of the PCs in VCC North or Pitts 4114, you will probably find this directory at either C:\Apps\orant or at C:\Apps1\orant. If you are using your own computer, then it will be where you installed Oracle. Remember the path name to the orant directory.

Now return to the Visual C++ window. On the left side of this window is a listing of the files in the ESQlProj.dsw project that you have opened. One of the Source files in the list is PROGRAM.PC. Click once on this file name to highlight it. Now click on the Project menu at the top of the Visual C++ window and select the Settings menu item in the menu that pops down. The Project Settings window appears on the screen. Click on the Custom Build tab on the right side of this window and look at the text in the textbox labeled Commands. This text is a file name that has Orant in the middle. Make sure that the beginning of this file name is the correct path name to the orant folder. If the path name is incorrect, edit it to correct it. Then click on the OK button at the bottom of the Project Settings window.

Back in the Visual C++ window, click once on the file name SQLLIB80.LIB on the left side of the window to highlight this file name. Now click on the View menu at the top of the Visual C++ window and then on the Properties menu item that appears in the menu that pops down. The Source File Properties window appears. Click on the General tab and look at the path name in the textbox labeled "File name:". If the path to the orant directory is incorrect, correct it in the lower textbox labeled "Persist as:". Then click on the close button for the window (the square with an x in it in the upper right corner of the window).

Now click on the Tools menu at the top of the Visual C++ window, and then on the Options menu item. In the Options window that pops up, click on the Directories tab. In the box labeled "Show directories for" select "Include files". You should see a gray box at the end of the list of

Directories. Click inside this gray box twice to activate it. In the box type

```
C:\Apps1\Orant\Pro80\C\Include
```

changing the path name to the orant folder as necessary. Then click on the OK button. At this point Visual C++ should be configured correctly to use the Pro*C precompiler.

4. Looking at the Program in the File movie.pc

Begin by studying the `Program.pc` program listed at the end of this document and displayed in the Visual C/C++ window. Execution of the program begins in the main function by prompting the user for the names of two actors. It then converts these actor names into actor IDs using two queries to the Actors relation in *The Movies Database*. It then creates a cursor for a query over the Casts and Films relations to get the titles of all films in which the two actors both played roles. As each such film title is retrieved using the cursor, the title is displayed on the screen.

All the C code required to implement the program is already present in the file `Program.pc`. To finish the program you must add the necessary embedded SQL. The places where this must be done are indicated by block comments. The next section provides the details that you will need to know to insert the required embedded SQL statements.

5. Completing the movie.pc Program

The comment blocks in the `Program.pc` program where code must be inserted are numbered. These numbers are used in the discussions below to refer to each comment block. The window displaying the program in Microsoft Visual C/C++ is a WYSIWYG editor window. You can use it to make the required additions to the program text using standard word processing techniques.

Comment Block #1

The first addition that you must make to the `Program.pc` program is to include the SQL communications area so that the program can communicate status information with Oracle. The name of the file that defines the SQL communications area is `sqlca.h`. The easiest way to include it in your program is to use an EXEC SQL INCLUDE statement. Thus, you need to add the line: `EXEC SQL INCLUDE "sqlca.h";` to the program.

Comment Block #2

Here you need to add the declarations for program variables that will be used in SQL queries. You must declare the following variables:

Variable Name	Variable Type	Initial Value (if any)
stagename1	varchar length 50	none
stagename2	varchar length50	none
filmname	varchar length 100	none
actorid1	int	none
actorid2	int	none
username	char length 10	your Oracle userid in " "
password	char length 15	your Oracle password in " "
db_string	char length 10	"ora8"

The general syntax to declare SQL variables is the following:

```
EXEC SQL BEGIN DECLARE SECTION;  
    type variable;          /* for int variables */
```

```

        type  variable[length]; /* for varchar and char */
        type  variable[length] = value; /* for initial values */
        .
        .
        .
EXEC SQL END DECLARE SECTION;

```

Add the necessary lines to declare the required SQL variables as described above. Remember that in C, when declaring an array to represent a string, the size of the array is specified in square brackets following the name of the variable as shown in the examples above.

Comment Block #3

You need to set up error handling so that if Oracle fails or if a query doesn't find any tuples that satisfy a query the program displays an appropriate message. This might occur, for example, if the user enters the name of an actor that is not in *The Movies Database*.

In the case of an Oracle error, you want to call the `sqlerror()` function that is defined at the end of the `Program.pc` file. This function consults the SQL communication area (`sqlca`) for a message about what is wrong. To do this, add the following line to the program:

```
EXEC SQL WHENEVER SQLERROR DO sqlerror();
```

In the case where a query doesn't find any tuples, you need to print an appropriate message. To do this, add the following line to the program:

```
EXEC SQL WHENEVER NOT FOUND DO printf("Not found.\n");
```

Comment Block #4

At this point, you want to connect to the Oracle database so that you can start to query it. Since it is possible that you might want to use more than one database, each database has to have a name. In this program, we will use only one database, but we still need to name it. We will name it `DB_NAME` using the following statement that you must add to the program:

```
EXEC SQL DECLARE DB_NAME DATABASE;
```

Now you must connect to the database that we want to call `DB_NAME`. Do this with the following command that you must add to the program:

```
EXEC SQL CONNECT :username IDENTIFIED BY :password
        AT DB_NAME USING :db_string;
```

Note that this command uses the program variables `username`, `password`, and `db_string` defined earlier.

Comment Block #5

Here you must insert two `SELECT-INTO-FROM-WHERE` statements that use *The Movies Database* to convert the two actor's names entered by the user (in variables `stagename1` and `stagename2`) to actor IDs that are put into variables `actorid1` and `actorid2`. The general form for this statement is the following:

```

SELECT <attribute list>
INTO <variable list>
FROM <relation list>
WHERE <condition>;

```

Remember that when a program variable is used in a `SELECT` statement, a colon must precede its name. Insert the required two `SELECT` statements into the program. Remember that you must append the prefix `"movies."` to the names of relations in *The Movies Database*.

Since you want to use database `DB_NAME` to process this query, precede your query with:

```
EXEC SQL AT DB_NAME
```

rather than just `EXEC SQL` like you have done before.

Comment Block #6

The query for comment block #6 computes the titles of all films in which both actors appeared. There may be more than one film title identified by this query. Hence, because of the multiple results, you must use a cursor for this query to retrieve the results one by one.

Here you must create the cursor and define the query for it. This query takes the actor IDs of the two actors (in variables `actorid1` and `actorid2`) and finds the titles of all movies in which they both appeared. The format of the SQL statement to create a cursor and define its query is the following:

```
EXEC SQL AT DB_NAME DECLARE <cursor name> CURSOR FOR
SELECT <attribute list>
FROM <relation list>
WHERE <condition>;
```

Design the required query and use it to declare a cursor named `filmcursor` in the program. Note again that you must use `AT DB_NAME` to specify the database to use for processing the query.

Comment Block #7

Once the cursor is defined, you need to open it. The general format of the SQL statement that opens a cursor is the following:

```
EXEC SQL OPEN <cursor name>;
```

Insert the SQL statement to open cursor `filmcursor`. Also remember to add `AT DB_NAME` to the statement to specify which database to use.

Comment Block #8

The program is about to enter a loop that will fetch film titles in the query result using the cursor. However, before entering the loop, you must create a way to terminate the loop when all film titles have been retrieved. The general form of the SQL statement to do this is the following:

```
EXEC SQL WHENEVER NOT FOUND DO <C statement>;
```

In this case, you want to break out of the loop that the program will be executing. Hence, the C statement that you want to execute is the `break` statement, which consists of the single word `break`.

Insert the necessary SQL statement to terminate the loop when all results have been fetched from the cursor. In this case you don't need to add `AT DB_NAME` since this statement is done locally in the program.

Comment Block #9

Inside the loop the program fetches the next film title in the query result. This is done using the `FETCH SQL` statement. The general format for this statement is the following:

```
EXEC SQL FETCH <cursor name> INTO <variable list>;
```

In this case, the cursor name is `filmcursor` and the query for cursor `filmcursor` has only one

attribute in its SELECT. Hence, there should be only one variable in the variable list of the FETCH statement. This variable is program variable `filename`. Write the required FETCH statement to fetch the film title from the query result. Remember that program variables must be preceded by a colon and that `AT DB_NAME` must be added since this fetch statement must be processed by a specific database.

Comment Block #10

At this point, the loop has terminated and all film titles have been displayed. You must now close the cursor. The format of the statement required to do this is:

```
EXEC SQL CLOSE <cursor name>;
```

Insert this statement into the program. Since the cursor is associated with a specific database, you must include `AT DB_NAME` in this statement.

Comment Block #11

Before terminating the program you should commit the transaction so that others can use the database. The format of the statement required to do this is:

```
EXEC SQL COMMIT WORK RELEASE;
```

While you are searching the database, you do not want other users to be making changes to it. By committing the transaction, you are telling Oracle that your search is done and that it is now safe for others to make changes to the database.

Insert this statement into the program and include `AT DB_NAME` since again the transaction you are committing is for a specific database.

At this point, the program is complete and you are ready to try compiling and running it.

6. Compiling and Running the Program

Now you are ready to compile and run the program. To make this as simple as possible, the Visual C/C++ workspace has been set up to do everything necessary to run the Pro*C preprocessor and then compile and link the program so that it is ready to run.

To preprocess, compile and link your program click on the Build button in the toolbar at the top of the Visual C/C++ window (see the picture below).



Any messages, including error messages, will appear in the windowpane at the bottom of the Visual C/C++ window. You can ignore most warning messages, but error messages must be fixed. If the final message in this window says that there are 0 errors, then you are ready to run the program and can skip the next few paragraphs. If there are error messages, then you will need to fix the program and try building it again.

If there are errors, scroll to the top of the message windowpane and find the first error message. Try to figure out what is wrong. Before you try to fix the error, notice which window you are editing. You can see this in the blue title bar at the top of the Visual C/C++ window. If the title bar has `Program.pc` in it, then go ahead and fix the problem. If the title bar has `Program.c` in it, then you must switch to file `Program.pc` before you fix the problem. To do this, look in the

windowpane to the left of the editor window. You should see Program.pc listed. Double click on it and the editor window will change to file Program.pc. Now you can go ahead and fix the error.

What is the file Program.c that might appear in the editor window? It is the output of the Pro*C preprocessor. It has all the SQL statements removed and replaced by function calls and other stuff. Making changes to this file will not correct the errors because this file is deleted and recreated the next time that you build the program. So be careful to make all your program fixes in the Program.pc file and not the Program.c file.

After you have fixed a few of the errors, you might want to Build the program again. Visual C/C++ tends to generate several error messages for the same mistake. One fix may get rid of several messages. The error messages are not always useful in identifying the problem. However, they do tell you where the problem is. Do your best to figure out what is wrong. If you get rid of the errors one by one, eventually the program will build error free.

To run the program once all the errors are fixed, click on the Execute button (it looks like an exclamation point, see the picture above). The program will run and prompt you for the names of two actors. You might try any of the following three pairs:

Humphrey Bogart	William Shatner	Woody Allen
Lauren Bacall	Leonard Nimoy	Diane Keaton

Each pair has appeared in several movies together. If the program doesn't run correctly, try to figure out what is wrong, fix it, and build and run it again, until you get it working correctly.

7. Things to Think About

This studio has only scratched the surface of what you can do with Embedded SQL using Oracle's Pro*C. For example, it is likely that much of the Student Information System is written using Pro*C. If you are interested, you might try modifying your program to do other things with *The Movies Database*.

If you are interested in exploring the game *Degrees of Kevin Bacon* in more detail, you might find the following web site of interest: <http://www.cs.virginia.edu/~bct7m/bacon.html> . For a web-based movies database see the following web site: <http://us.imdb.com/> .

In the next studio you will learn how to use the Oracle forms package to build a graphical application interface much like you did with Microsoft Access.

9. File Program.pc

```
/*-----*/
/*
/* Modify this program by adding the necessary code as instructed */
/* at each comment block starting with: */
/*
/*          ***** Begin *****
/*
/* and ending with:
/*
/*          ***** End *****
/*-----*/
```

```

/* Include header files */
#include <stdio.h>
#include <string.h>

// prototype for error functions
void sqlerror();
void exit(int);

/***** Begin *****/
/*-----*/
/* 1. */
/* Include the SQL communication Area. */
/*-----*/
/***** End *****/

int main()
{
    char name1[25];
    char name2[25];

    /***** Begin *****/
    /*-----*/
    /* 2. */
    /* Declare variables for use in SQL queries */
    /* stageName1 varchar length 50 */
    /* stageName2 varchar length 50 */
    /* filmName varchar length 100 */
    /* actorid1 int */
    /* actorid2 int */
    /* Declare variables to connect to database */
    /* username char length 10, init to Oracle userid */
    /* password char length 15, init to Oracle password*/
    /* db_string char length 20, init to "ora8" */
    /*-----*/
    /***** End *****/

    /***** Begin *****/
    /*-----*/
    /* 3. */
    /* Setup error handling for cases when Oracle */
    /* or a query fails. */
    /*-----*/
    /***** End *****/

    /***** Begin *****/
    /*-----*/
    /* 4. */
    /* Connect to the Oracle database using name DB_NAME */
    /*-----*/
    /***** End *****/

```

```

/* get actors stagenames */
printf("Enter the name of first actor: ");
gets(name1);
strcpy(stagename1.arr, name1);
stagename1.len = strlen(stagename1.arr);

printf("Enter the name of second actor: ");
gets(name2);
strcpy(stagename2.arr, name2);
stagename2.len = strlen(stagename2.arr);

/***** Begin *****/
/*-----*/
/* 5. */
/* Embedded SELECT statements to get the */
/* actor IDs for the given actor's */
/* */
/*-----*/
/***** End *****/
/***** Begin *****/
/*-----*/
/* 6. */
/* Embedded SELECT to create a cursor named 'filmcursor' that */
/* selects the list of degree 0 actors for the actor whose ID */
/* is in variable 'actorId'. */
/* */
/*-----*/
/***** End *****/

/***** Begin *****/
/*-----*/
/* 7. */
/* Embedded SQL to open cursor 'filmcursor'. */
/* */
/*-----*/
/***** End *****/

/***** Begin *****/
/*-----*/
/* 8. */
/* Embedded SQL to break from the loop below when there are */
/* no more tuples to fetch for cursor 'filmcursor'. */
/* */
/*-----*/
/***** End *****/

/* loop to fetch tuples from the database */
printf("\n\nJoint Films for %s and %s\n", stagename1.arr,
stagename2.arr);
printf("-----\n");
for (;;) {

    /***** Begin *****/
    /*-----*/
    /* 9. */
    /* Embedded SQL to fetch the next tuple for cursor */
    /* */

```

```

        /* 'filmcursor' into variable 'filename'. */
        /*-----*/
        /****** End ******/

        /* put null terminator at end of filename */
        filename.arr[filename.len] = '\0';

        /* print film name */
        printf("%s\n", filename.arr);
    }

    /****** Begin ******/
    /*-----*/
    /* 10. */
    /* Embedded SQL to close cursor 'filmcursor' */
    /*-----*/
    /****** End ******/

    /****** Begin ******/
    /*-----*/
    /* 11. */
    /* Commit the transaction and disconnect from the dbms. */
    /*-----*/
    /****** End ******/

    printf("\n\n");
    return (0);
}

void sqlerror()
{
    printf("\n\nOracle Error...\n");
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
    exit(1);
}

```