CSC343 Tutorial - Week 9

TA: Lei Jiang

This Week:

Embedded SQL
(http://www.cs.toronto.edu/%7Elibkin/csc343/f02/embedded/embedsql.html)
Test 1 back (after tutorial)

What's Next:

10 March 9 Test2
11 March 16 Tutorial
12 March 23 Tutorial
13 March 30 Tutorial
14 April 5 In-class Final

Static Embedded SQL

```
/*****************************************************
**
** A sample program that demonstrates the use of Static embedded SQL.
** Before compiling this program, be sure you have created a table
** called video and inserted some tuples in it.
**
*****************************************************/
#include <stdio.h>

/* sqlca: is the sql communications area. All error codes
   are returned from db2 in that structure which is filled
   each time an interaction with db2 takes place.
*/

EXEC SQL INCLUDE SQLCA;          /* SQL communication area structure */

EXEC SQL BEGIN DECLARE SECTION;  /* declare host variables    */
 char db_name[8];                /* database name             */
 char video_title[30];           /* title of the video        */
 short video_id;                 /* serial number             */
 char director[20];              /* director name             */
EXEC SQL END DECLARE SECTION;

/* These lines are redundant here because the default
   action is to continue. They just show the kind of
   errors that could arise and one way to control them.
*/

EXEC SQL WHENEVER SQLWARNING CONTINUE;  /* sqlca.sqlcode > 0    */
EXEC SQL WHENEVER SQLERROR CONTINUE;    /* sqlca.sqlcode < 0    */
EXEC SQL WHENEVER NOT FOUND CONTINUE;   /* sqlca.sqlcode = 100  */
                                        /* sqlca.sqlcode = 0 (no error)
*/

void main() {

strcpy(db_name, "csc343h");

/* C variables are preceded by a colon when they are passed to DB2 */

EXEC SQL CONNECT TO :db_name;

if (sqlca.sqlcode != 0) {
        printf("Connect failed!: reason %ld\n", sqlca.sqlcode);
        exit(1);
}

/* cursor delcaration. Have to declare a cursor each time you
   want tuples back from db2
*/

EXEC SQL DECLARE c1 CURSOR FOR
        SELECT video_title
        FROM video;

/* you have to open the cursor in order to get tuples back */

EXEC SQL OPEN c1;

do {

  /* fetch tuples from the cursor. This will execute the statement
     the cursor implements and will return the results */

  EXEC SQL FETCH c1 into :video_title;
  if (SQLCODE != 0) break;     /* SQLCODE refers to sqlca.sqlcode */

  /* host variables should have ':' prefix when they are used in DB2
commands */

  printf("%s\n",video_title);

} while (1);
EXEC SQL CLOSE c1;
EXEC SQL CONNECT RESET;
}
```

Something to be aware of …

The use of SQL command within a host language (e.g. C) program is called **Embedded SQL**. Details of embedded SQL also depend on the host language. The syntax sometimes varies.

1. **SQL statements must be marked clearly;**

   Example: insert a row into Table Sailors based on the values in defined variables.

   EXEC SQL *INSERT INTO Sailors VALUES (:c_sid, :c_sname, :c_rating, :c_age)*

2. **Any host language variable used to pass arguments into an SQL command must be declared in SQL.**

   Example:

   ```
   EXEC SQL BEGIN DECLARE SECTION
           long c_sid;
           char c_sname[20];
           short c_rating;
           float c_age;
           short c_para;
   EXEC SQL BEGIN DECLARE SECTION
   ```

3. **SQLCODE is used to store error code returned from db2; need to check it after every interaction with the database.**
   - record not found: ***sqlca.sqlcode == 100***
   - no error: ***sqlca.sqlcode == 0***
   - error or warning: ***sqlca.sqlcode != 0***

   or,

   EXEC SQL WHENEVER [SQLERROR|NOT FOUND][CONTINUE|GOTO]

4. **SQL commands deal with tables, while the interface to the host language is constrained to one row at a time.**

   Example:

   ```
   EXEC SQL SELECT S.sname, S.age
                   INTO :c_sname, :c_age
                   FROM Sailors S WHERE S.sid = :c_sid;
   ```

   Given a *c_sid* (e.g. 29), this statement will return this sailor's name and age to variable *c_sname* (e.g. Brutus) and *c_age* (e.g. 33) if this *c_sid* exists.

   Example:

   ```
   EXEC SQL SELECT S.sname, S.age
                   INTO :c_sname, :c_age
                   FROM Sailors S
                   WHERE S.rating > :c_para;
   ```

   Given a *c_para* (e.g. 5), the SQL command may return more than one row. But the INTO clause is not adequate to support collections of rows. Host variables *c_sname* and *c_age* may just return the first row of the result set.

5. Cursor

We can declare a cursor on any relation or on any SQL query.
Once a cursor is **declare**d, we can

1. **open** it (which positions the cursor just before the first row);
2. **fetch** the next row;
3. **move** the cursor (to the next row, to the row after the next *n*, to the first row, or to the previous row etc. by specifying additional parameters for the FETCH command);
4. or **close** the cursor.

Example:

```
DECLARE sinfo CURSOR FOR
SELECT S.sname, S.age
FROM Sailors S
WHERE S.rating > :c_para;

OPEN sinfo;

FETCH sinfo INTO :c_sname, :c_age;
Do While (SQLSTATE <> '02000') // SQLSTATE == '02000' denotes NO DATA
    //deal with the current row …..…
    FETCH sinfo INTO :c_sname, :c_age;
End While
CLOSE sinfo;
```

6. Variations of Cursor

The general formal of a cursor declaration is:

```
DECLARE cursorname [INSENSITIVE][SCROLL] CURSOR FOR
some query
[ORDER BY order-item-list]
[FOR READ ONLY | FOR UPDATE]
```

A cursor can be declared to be **a read-only cursor**, or, if it is a cursor on a base relation or an updateable view, to be **an updateable cursor**.
(see example below)

Dynamic Embedded SQL

```
/*****************************************************
**
** A sample program that demonstrates the use of Dynamic embedded SQL.
** Before compiling this program, be sure you have created a table
** called video and inserted some tuples in it.
**
*****************************************************/
#include <stdio.h>

EXEC SQL INCLUDE SQLCA;

/* The next statement is used in dynamic SQL programs */
/* where the output (the number of columns or their   */
/* types) is not fixed in the compile time. Here we   */
/* don't need it, since the output is fixed.          */
/*                                                    */
/* EXEC SQL INCLUDE SQLDA;                            */

EXEC SQL BEGIN DECLARE SECTION;  /* declare host variables    */
 char  db_name[8];
 char  qstring[100];             /* buffer for SQL query      */
 char  video_title[30];
 short video_id;
 char  director[20];
EXEC SQL END DECLARE SECTION;

void main() {

strcpy(db_name, "csc343h");

EXEC SQL CONNECT TO :db_name;

if (SQLCODE != 0) {
        printf("Connect failed!: reason %ld\n", sqlca.sqlcode);
        exit(1);
}

/* construct a query */
strcpy(qstring, "SELECT video_title FROM video");

/* Prepare the query in the run time */
EXEC SQL PREPARE Q1 FROM :qstring;

if (SQLCODE != 0) {
        printf("PREPARE failed!: reason %ld\n", sqlca.sqlcode);
        exit(1);
}

EXEC SQL DECLARE c1 CURSOR FOR Q1;

EXEC SQL OPEN c1;

do {

  EXEC SQL FETCH c1 into :video_title;
  if (SQLCODE != 0) break;

  printf("%s\n",video_title);

} while (1);

EXEC SQL CLOSE c1;
EXEC SQL CONNECT RESET;
}
```

More Dynamic Embedded SQL

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

EXEC SQL INCLUDE SQLCA;

#define CHECKERR(CE_STR)   if (sqlca.sqlcode != 0) {printf("%s failed.
Reason %ld\n", CE_STR, sqlca.sqlcode); exit(1); }

int main(int argc, char *argv[]) {

   EXEC SQL BEGIN DECLARE SECTION;
      char  table_name[19];
      char  st[80];  (1)
      char  parm_var[19];
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: DYNAMIC\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;
          CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: dynamic [userid passwd]\n\n");
      return 1;
   } /* endif */

   strcpy( st, "SELECT tabname FROM syscat.tables" );
   strcat( st, " WHERE tabname <> ?" );
   EXEC SQL PREPARE s1 FROM :st;  (2)
   CHECKERR ("PREPARE");

   EXEC SQL DECLARE c1 CURSOR FOR s1;  (3)

   strcpy( parm_var, "STAFF" );
   EXEC SQL OPEN c1 USING :parm_var;  (4)
   CHECKERR ("OPEN");
   do {
      EXEC SQL FETCH c1 INTO :table_name;  (5)
      if (SQLCODE != 0) break;

      printf( "Table = %s\n", table_name );
   } while ( 1 );

   EXEC SQL CLOSE c1;  (6)
   CHECKERR ("CLOSE");

   EXEC SQL COMMIT;
   CHECKERR ("COMMIT");

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : DYNAMIC.SQ
```

Updateable Cursor

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

EXEC SQL INCLUDE SQLCA;

#define  CHECKERR(CE_STR)   if (sqlca.sqlcode != 0) {printf("%s failed.
Reason %ld\n", CE_STR, sqlca.sqlcode); exit(1); }

int main(int argc, char *argv[]) {

   EXEC SQL BEGIN DECLARE SECTION;
      char   pname[10];
      short  dept;
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: OPENFTCH\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: openftch [userid passwd]\n\n");
      return 1;
   } /* endif */

   EXEC SQL DECLARE c1 CURSOR FOR  (1)
            SELECT name, dept FROM staff WHERE job='Mgr'
            FOR UPDATE OF job;

   EXEC SQL OPEN c1;  (2)
   CHECKERR ("OPEN CURSOR");

   do {
      EXEC SQL FETCH c1 INTO :pname, :dept;  (3)
      if (SQLCODE != 0) break;

      if (dept > 40) {
         printf( "%-10.10s in dept. %2d will be demoted to Clerk\n",
            pname, dept );
         EXEC SQL UPDATE staff SET job = 'Clerk'  (4)
            WHERE CURRENT OF c1;
         CHECKERR ("UPDATE STAFF");
      } else {
         printf ("%-10.10s in dept. %2d will be DELETED!\n",
            pname, dept);
         EXEC SQL DELETE FROM staff WHERE CURRENT OF c1;
         CHECKERR ("DELETE");
      } /* endif */
   } while ( 1 );

   EXEC SQL CLOSE c1;  (5)
   CHECKERR ("CLOSE CURSOR");

   EXEC SQL ROLLBACK;
   CHECKERR ("ROLLBACK");
   printf( "\nOn second thought -- changes rolled back.\n" );

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : OPENFTCH.SQC */
```