CSC343 Worksheet 8 Solution

June 24, 2020

1. a)

```
#include sqlcli.h
void askUserForPrice() {
    SQLHENV myEnv;
    SQLHDBC myCon;
    SQLHSTMT execStat;
    SQLINTEGER worth, worthInfo;
}

```

Notes:

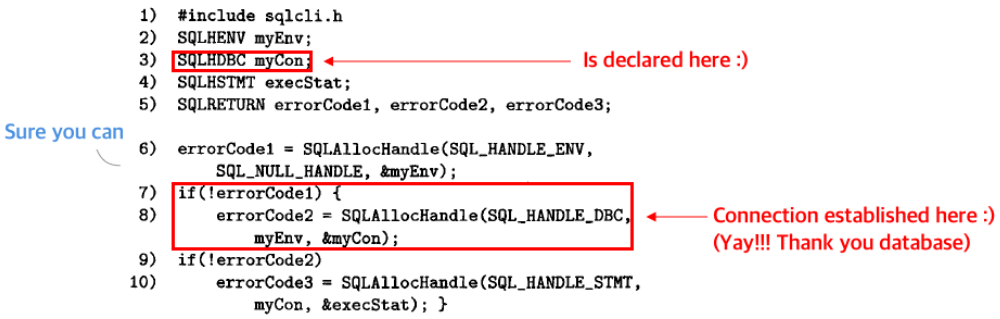
- Using Call-Level Interface
  - Uses host language to connect to and access a database
  - Replaces embedded SQL
- Standard SQL/CLI
  - Is database CLI for C
  - Included in file *sqlcli.h*
  - Creates deals with four kinds of records

    1. Environment handle
       * Prepares one or more connections to database server
       * Is required
       * Is allocated using **SQLHENV**
       * Is established via function **SQLAllocHandle**

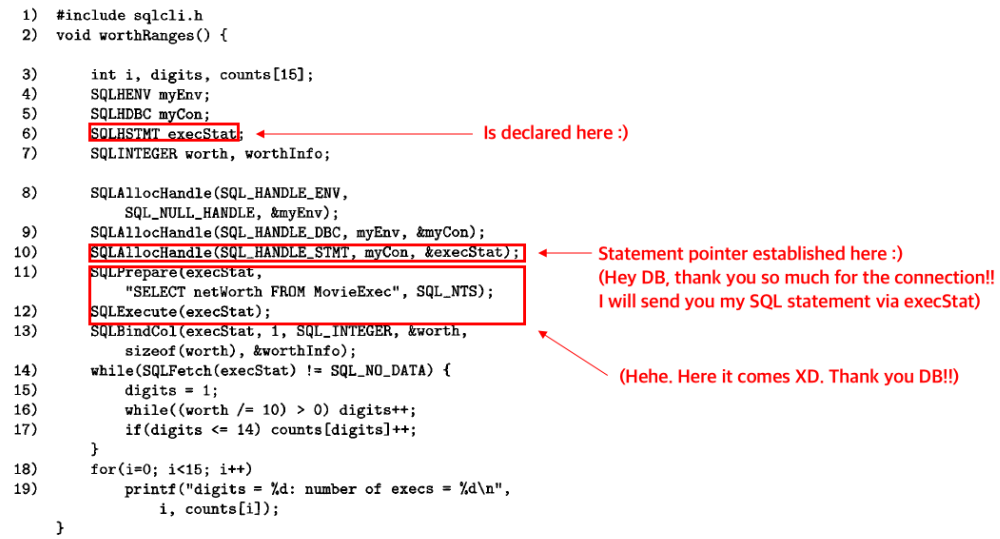
```
1)  #include sqlcli.h
2)  SQLHENV myEnv;
3)  SQLHDBC myCon;          <-- Is declared here :)
4)  SQLHSTMT execStat;
5)  SQLRETURN errorCode1, errorCode2, errorCode3;

6)  errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,     <-- Connection is prepared here :)
        SQL_NULL_HANDLE, &myEnv);                       (Hey DB, can I connect with you?)
7)  if(!errorCode1) {
8)      errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC,
            myEnv, &myCon);
9)  if(!errorCode2)
10)     errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT,
            myCon, &execStat); }
```

2. Connection handle
   * Conenects application program to database
   * Is required
   * Is declared after **SQLHENV**
   * Is allocated using **SQLHDBC**
   * Is established via function **SQLAllocHandle**



3. Statements
   * Created by application program (the user)
   * Can be created as many as needed
   * Holds information about a single SQL statement, including cursor
   * Can represent different SQL statements at different times
   * Is required
   * Is declared after **SQLHDBC**
   * Is allocated using **SQLHSTMT**
   * Is sent using the function **SQLAllocHandle**



4. Descriptions
   * Holds information about either tuples or parameters
   * Each statement has this information implicitly

- Processing Statements
  - is done using **SQLPrepare** and **SQLExecute**

$$\textbf{SQLPrepare}(sh, st, SQL\_NTS) \tag{1}$$
$$\textbf{SQLExecute}(sh) \tag{2}$$

  - sh is the statement handle created using **SQLHSTMT**
  - SQL_NTS evaluates the length of string in st

    Example:

```
1    SQLPrepare(execStat, "SELECT netWorth FROM MovieExec",
     SQL_NTS);
2    SQLExecute(execStat);
3
```

  - the function **SQLExecDirect** combines **SQLPrepare** and **SQLExecute**

    Example 2:

```
1    SQLExecDirect(execStat, "SELECT netWorth FROM MovieExec",
     SQL_NTS);
2
```

- Fetching Data From
  - Fetch
    - **Syntax:** SQLFetch(sh)
    - Executes statement in **SQLPrepare** and **SQLExecute** and stores result to variable in **SQLBindCol**
    - Fetches a row per call
    - Returns a value of type **SQLRETURN**, indicating either success or error
  - SQLBindCol
    - **Syntax:** SQLBindCol($sh, colNo, colType, pVar, varSize, varInfo$)
      - **sh**: the handle of statement (e.g execStat)
      - **colNo**: the position of column in tuple we obtain
      - **colType**: the SQL data type of variable (e.g. SQL_INTEGER, SQL_CHAR)
      - **pVar**: the pointer to variable the value is placed
      - **varSize**: the length in bytes of the value in $pVar$
      - **varInfo**: a pointer to an integer used by SQLBindCol for additional value about the value produced
    - Stores data from **SQLFetch** to host-language variable
    - Must be setup before SQLFetch(sh) is run

```
1)  #include sqlcli.h
2)  void worthRanges() {

3)      int i, digits, counts[15];
4)      SQLHENV myEnv;
5)      SQLHDBC myCon;
6)      SQLHSTMT execStat;
7)      SQLINTEGER worth, worthInfo;

8)      SQLAllocHandle(SQL_HANDLE_ENV,
            SQL_NULL_HANDLE, &myEnv);
9)      SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);
10)     SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);
11)     SQLPrepare(execStat,
            "SELECT netWorth FROM MovieExec", SQL_NTS);
12)     SQLExecute(execStat);
13)     SQLBindCol(execStat, 1, SQL_INTEGER, &worth,
            sizeof(worth), &worthInfo);
14)     while(SQLFetch(execStat) != SQL_NO_DATA) {
15)         digits = 1;
16)         while((worth /= 10) > 0) digits++;
17)         if(digits <= 14) counts[digits]++;
        }
18)     for(i=0; i<15; i++)
19)         printf("digits = %d: number of execs = %d\n",
                i, counts[i]);
    }
```

The value to fetch is defined here :)

The storage location is defined here :)
(Hey DB, when data is fetched, could you
store the fetched value
of SQL_INTEGER datatype to
worth variable? Here is the address)

Value is fetched here :)