CSC343 Worksheet 8 Solution

June 24, 2020

1. a)

```
 #include <float.h>

 #include sqlcli.h

 void askUserForPrice() {

     float targetPrice, minDiff, speedSol, minDiff = FLT_MAX;
     int modelSol;
     char makerSol;

     SQLHENV myEnv;
     SQLHDBC myCon;
     SQLHSTMT execStat;

     SQLINTEGER model, modelInfo, speedInfo, ram, ramInfo, hd,
hdInfo, priceInfo, makerInfo;
     SQLREAL speed, price;
     SQLCHAR maker;


     errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
                  SQL_NULL_HANDLE, &myEnv);

     if (!errorCode1) {
         errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon
);
     }

     if (!errorCode2) {
         errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
execStat)
     }

     if (!errorCode3) {
         SQLPrepare(execStat,
                   "SELECT model, speed, ram, hd, price, maker "
                   "FROM Product NATURAL JOIN PC", SQL_NTS);
         SQLExecute(execStat);
```

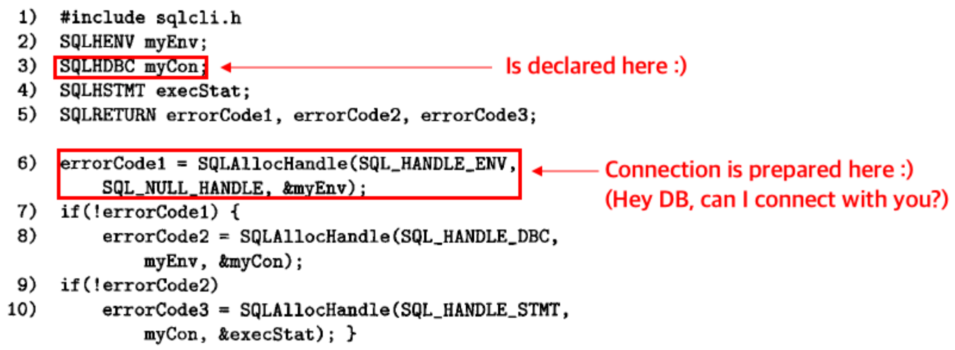
```c
         SQLBindCol(execStat, 1, SQL_INTEGER, &model, sizeof(model), &modelInfo);
         SQLBindCol(execStat, 2, SQL_FLOAT, &speed, sizeof(speed), &speedInfo);
         SQLBindCol(execStat, 3, SQL_INTEGER, &ram, sizeof(ram), &ramInfo);
         SQLBindCol(execStat, 4, SQL_INTEGER, &hd, sizeof(hd), &hdInfo);
         SQLBindCol(execStat, 5, SQL_FLOAT, &price, sizeof(price), &priceInfo);
         SQLBindCol(execStat, 5, SQL_CHAR, &maker, sizeof(maker), &makerInfo);

         printf("Enter target price:");
         scanf("%f", &targetPrice);

         while (SQLFetch(execStat) != SQL_NO_DATA) {

             if (abs(price - targetPrice) >= minDiff) {
                 continue;
             }

             minDiff = abs(price - targetPrice);
             modelSol = model;
             speedSol = speed;
             makerSol = maker;
         }

         printf("maker=%c, model=%d, speed=%.2f\n", makerSol, modelSol, speedSol);

     }
 }

```

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**

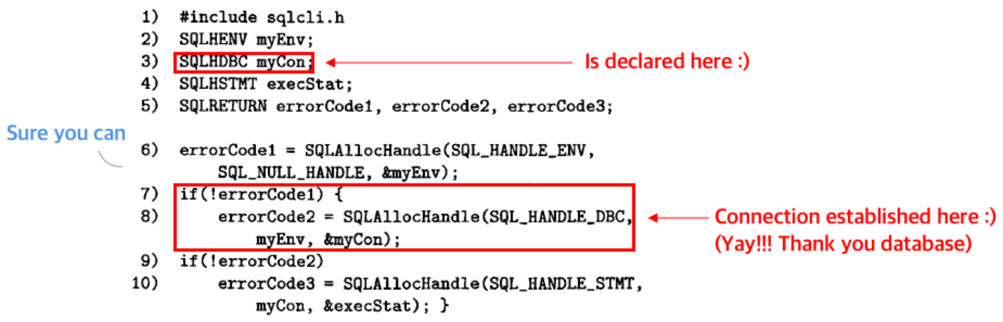
* Is established via function **SQLAllocHandle**



```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC myCon;
SQLHSTMT execStat;
SQLRETURN errorCode1, errorCode2, errorCode3;

errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
    SQL_NULL_HANDLE, &myEnv);
if(!errorCode1) {
    errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC,
        myEnv, &myCon);
if(!errorCode2)
    errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT,
        myCon, &execStat); }
```

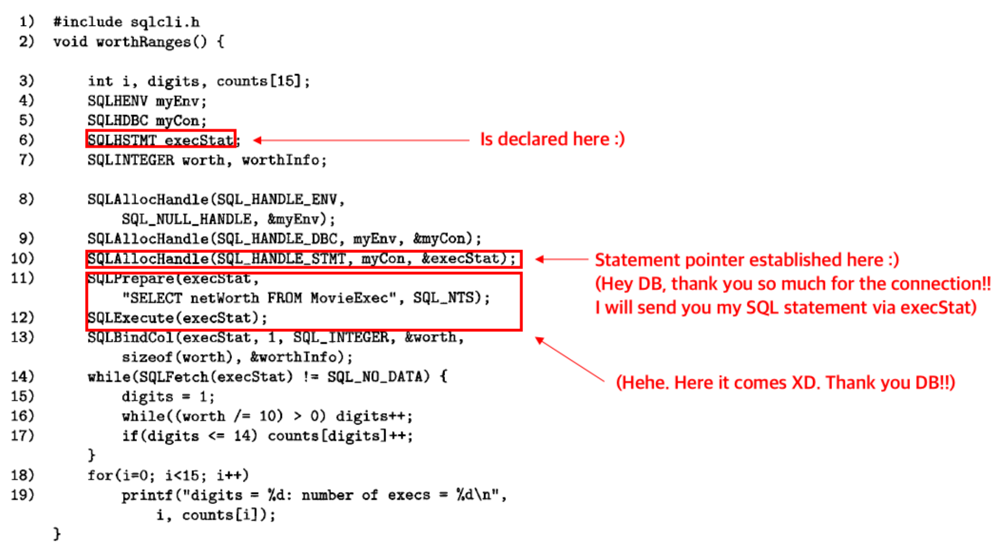
Is declared here :)
Connection is prepared here :)
(Hey DB, can I connect with you?)

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



```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC myCon;
SQLHSTMT execStat;
SQLRETURN errorCode1, errorCode2, errorCode3;

errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
    SQL_NULL_HANDLE, &myEnv);
if(!errorCode1) {
    errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC,
        myEnv, &myCon);
if(!errorCode2)
    errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT,
        myCon, &execStat); }
```

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**



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

    int i, digits, counts[15];
    SQLHENV myEnv;
    SQLHDBC myCon;
    SQLHSTMT execStat;
    SQLINTEGER worth, worthInfo;

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

Is declared here :)

Statement pointer established here :)
(Hey DB, thank you so much for the connection!!
I will send you my SQL statement via execStat)

(Hehe. Here it comes XD. Thank you DB!!)

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:

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

```

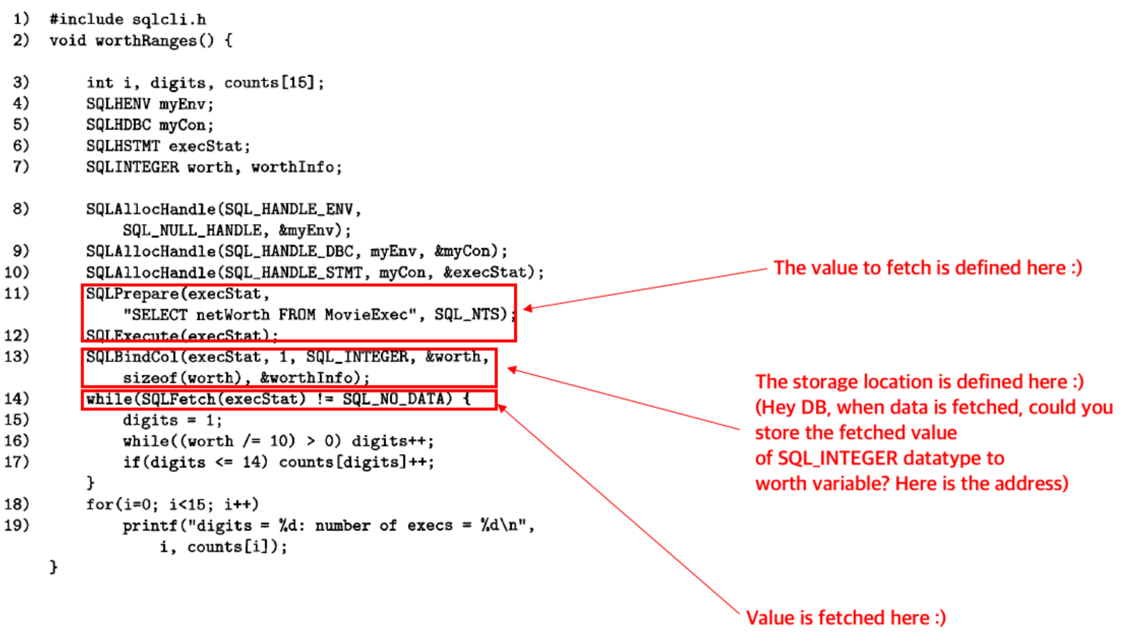
  - the function **SQLExecDirect** combines **SQLPrepare** and **SQLExecute**

  Example 2:

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

```

- 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



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

    int i, digits, counts[15];
    SQLHENV myEnv;
    SQLHDBC myCon;
    SQLHSTMT execStat;
    SQLINTEGER worth, worthInfo;

    SQLAllocHandle(SQL_HANDLE_ENV,
        SQL_NULL_HANDLE, &myEnv);
    SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);
    SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);
    SQLPrepare(execStat,
        "SELECT netWorth FROM MovieExec", SQL_NTS);
    SQLExecute(execStat);
    SQLBindCol(execStat, 1, SQL_INTEGER, &worth,
        sizeof(worth), &worthInfo);
    while(SQLFetch(execStat) != SQL_NO_DATA) {
        digits = 1;
        while((worth /= 10) > 0) digits++;
        if(digits <= 14) counts[digits]++;
    }
    for(i=0; i<15; i++)
        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 :)