CSC343 Worksheet 8 Solution

June 24, 2020

1. a)

```c
#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);
```

1

```
36            SQLBindCol(execStat, 1, SQL_INTEGER, &model, sizeof(model
   ), &modelInfo);
37            SQLBindCol(execStat, 2, SQL_FLOAT, &speed, sizeof(speed),
    &speedInfo);
38            SQLBindCol(execStat, 3, SQL_INTEGER, &ram, sizeof(ram), &
   ramInfo);
39            SQLBindCol(execStat, 4, SQL_INTEGER, &hd, sizeof(hd), &
   hdInfo);
40            SQLBindCol(execStat, 5, SQL_FLOAT, &price, sizeof(price),
    &priceInfo);
41            SQLBindCol(execStat, 6, SQL_CHAR, &maker, sizeof(maker),
   &makerInfo);
42
43            printf("Enter target price:");
44            scanf("%f", &targetPrice);
45
46            while (SQLFetch(execStat) != SQL_NO_DATA) {
47
48                if (abs(price - targetPrice) >= minDiff) {
49                    continue;
50                }
51
52                minDiff = abs(price - targetPrice);
53                modelSol = model;
54                speedSol = speed;
55                makerSol = maker;
56            }
57
58            printf("maker=%c, model=%d, speed=%.2f\n", makerSol,
   modelSol, speedSol);
59
60        }
61    }
62
```

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**

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

Sure you can
6)   errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
         SQL_NULL_HANDLE, &myEnv);
7)   if(!errorCode1) {
8)       errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC,   ←——  Connection established here :)
             myEnv, &myCon);                                 (Yay!!! Thank you database)
9)   if(!errorCode2)
10)      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**

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

3)       int i, digits, counts[15];
4)       SQLHENV myEnv;
5)       SQLHDBC myCon;
6)       SQLHSTMT execStat;              ◄———— Is declared here :)
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);   ◄——  Statement pointer established here :)
11)      SQLPrepare(execStat,                                      (Hey DB, thank you so much for the connection!!
             "SELECT netWorth FROM MovieExec", SQL_NTS);          I will send you my SQL statement via execStat)
12)      SQLExecute(execStat);
13)      SQLBindCol(execStat, 1, SQL_INTEGER, &worth,
             sizeof(worth), &worthInfo);
14)      while(SQLFetch(execStat) != SQL_NO_DATA) {               (Hehe. Here it comes XD. Thank you DB!!)
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]);
     }
```

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

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

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

  - 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 :)

b)
```
1   #include sqlcli.h
2
3   void findLaptops() {
4
5       float minSpeed, minPrice;
6       int minRam, minHd;
7
8       SQLINTEGER model, modelInfo, speedInfo, ram, ramInfo, hd,
    hdInfo, priceInfo, makerInfo, screen, screenInfo;
9       SQLREAL speed, price;
10      SQLCHAR maker;
11
12      errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
```

```
13                        SQL_NULL_HANDLE , &myEnv );
14
15         if (!errorCode1) {
16               errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC , myEnv , &myCon
     );
17         }
18
19         if (!errorCode2) {
20               errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT , myCon , &
     execStat )
21         }
22
23         if (!errorCode3) {
24             SQLPrepare(execStat ,
25                       "SELECT model , speed , ram , hd , screen , price ,
     maker "
26                       "FROM Product NATURAL JOIN Laptop", SQL_NTS );
27             SQLExecute(execStat );
28             SQLBindCol(execStat , 1, SQL_INTEGER , &model , sizeof(model
     ), &modelInfo );
29             SQLBindCol(execStat , 2, SQL_FLOAT , &speed , sizeof(speed),
      &speedInfo );
30             SQLBindCol(execStat , 3, SQL_INTEGER , &ram , sizeof(ram), &
     ramInfo );
31             SQLBindCol(execStat , 4, SQL_INTEGER , &hd , sizeof(hd), &
     hdInfo );
32             SQLBindCol(execStat , 5, SQL_INTEGER , &screen , sizeof(
     screen), &screenInfo );
33             SQLBindCol(execStat , 6, SQL_FLOAT , &price , sizeof(price),
      &priceInfo );
34             SQLBindCol(execStat , 7, SQL_CHAR , &maker , sizeof(maker),
     &makerInfo );
35
36             printf("Enter minimum speed:");
37             scanf("%f", &minSpeed );
38
39             printf("Enter minimum ram:");
40             scanf("%f", &minRam );
41
42             printf("Enter minimum hard-drive space:");
43             scanf("%f", &minHd );
44
45             printf("Enter minimum price:");
46             scanf("%f", &minPrice );
47
48             while(SQLFetch(execStat) != SQL_NO_DATA) {
49                 if (
50                     speed >= minSpeed &&
51                     ram >= minRam &&
52                     hd >= minHd &&
53                     screen >= minScreen
54                 ) {
55                     printf("model=%d, speed=%.2f, ram=%d, hd=%d,
     screen=%d, price=%.2f, maker=%c",
```

```
56                              model , speed , ram , hd , screen , price , maker );
57                   }
58               }
59           }
60       }
61
```

c)
```
1      #include <stdbool.h>
2      #include <string.h>
3      ...
4      void printSpecifications () {
5          char targetMaker ;
6
7          SQLHENV myEnv ;
8          SQLHDBC myCon ;
9          SQLHSTMT execStat , subExecStat ;
10
11         SQLINTEGER model , modelInfo , speedInfo , ram , ramInfo , hd ,
     hdInfo , priceInfo , makerInfo , screen , screenInfo , color , colorInfo
     , printTypeInfo ;
12         SQLREAL speed , price ;
13         SQLCHAR maker , printType [50];
14
15         SQLRETURN errorCode1 , errorCode2 , errorCode3 ;
16
17         errorCode1 = SQLAllocHandle ( SQL_HANDLE_ENV ,
18                      SQL_NULL_HANDLE , &myEnv );
19
20         if (! errorCode1 ) {
21             errorCode2 = SQLAllocHandle ( SQL_HANDLE_DBC , myEnv , &myCon
     );
22         }
23
24         if (! errorCode2 ) {
25             errorCode3 = SQLAllocHandle ( SQL_HANDLE_STMT , myCon , &
     execStat );
26             errorCode4 = SQLAllocHandle ( SQL_HANDLE_STMT , myCon , &
     subExecStat );
27         }
28
29         if (! errorCode3 && ! errorCode4 ) {
30             printf (" Enter manufacturer :");
31             scanf ("%c", &targetMaker );
32
33             SQLPrepare ( execStat ,
34                     " SELECT maker , productType FROM Product "
35                     " GROUP BY maker , productType "
36                     " WHERE maker = ?", SQL_NTS );
37                 SQLBindParameter ( execStat , 1, ..., targetMaker , ...);
38             SQLExecute ( execStat );
39             SQLBindCol ( execStat , 1, SQL_CHAR , &maker , sizeof ( maker ),
     &makerInfo );
40             SQLBindCol ( execStat , 2, SQL_CHAR , &productType , sizeof (
     productType ), &productTypeInfo );
```

7

```
41
42            while (SQLFetch(execStat) != SQL_NO_DATA) {
43                if (strcmp(productType,'pc')) {
44                    SQLPrepare(subExecStat,
45                              "SELECT speed, ram, hd, price FROM PC
   "
46                              "NATURAL JOIN Product "
47                              "WHERE type= ?", SQL_NTS);
48                    SQLBindParameter(subExecStat, 1, ...,
   productType, ...);
49                    SQLExecute(subExecStat);
50
51                    SQLBindCol(subExecStat, 1, SQL_FLOAT, &speed,
   sizeof(speed), &speedInfo);
52                    SQLBindCol(subExecStat, 2, SQL_INTEGER, &ram,
   sizeof(ram), &ramInfo);
53                    SQLBindCol(subExecStat, 3, SQL_INTEGER, &hd,
   sizeof(hd), &hdInfo);
54                    SQLBindCol(subExecStat, 4, SQL_FLOAT, &price,
   sizeof(price), &priceInfo);
55
56                    while(SQLFetch(subExecStat) != SQL_NO_DATA) {
57                        printf("model=%d, speed=%.2f, ram=%d, hd=%d,
   price=%.2f, maker=%c, type=%s",
58                        model, speed, ram, hd, screen, price, maker,
   productType);
59                    }
60                } else if (strcmp(productType, 'laptop')) {
61
62                    SQLPrepare(subExecStat,
63                              "SELECT speed, ram, hd, screen, price
    FROM Laptop "
64                              "NATURAL JOIN Product "
65                              "WHERE type= ?", SQL_NTS);
66                    SQLBindParameter(subExecStat, 1, ...,
   productType, ...);
67                    SQLExecute(subExecStat);
68
69                    SQLBindCol(subExecStat, 1, SQL_FLOAT, &speed,
   sizeof(speed), &speedInfo);
70                    SQLBindCol(subExecStat, 2, SQL_INTEGER, &ram,
   sizeof(ram), &ramInfo);
71                    SQLBindCol(subExecStat, 3, SQL_INTEGER, &hd,
   sizeof(hd), &hdInfo);
72                    SQLBindCol(subExecStat, 4, SQL_INTEGER, &screen,
   sizeof(screen), &screenInfo);
73                    SQLBindCol(subExecStat, 5, SQL_FLOAT, &price,
   sizeof(price), &priceInfo);
74
75                    while(SQLFetch(subExecStat) != SQL_NO_DATA) {
76                        printf("model=%d, speed=%.2f, ram=%d, hd=%d,
   screen=%d, price=%.2f, maker=%c, type=%s",
77                        model, speed, ram, hd, screen, screen, price,
    maker, productType);
```

```
78                              }
79                      } else if (strcmp(productType, 'printer')) {
80                          SQLPrepare(subExecStat,
81                                      "SELECT color, printType, price FROM
    Printer "
82                                      "NATURAL JOIN Product "
83                                      "WHERE type= ?", SQL_NTS);
84                              SQLBindParameter(subExecStat, 1, ...,
    productType, ...);
85                          SQLExecute(subExecStat);
86
87                          SQLBindCol(subExecStat, 1, SQL_INTEGER, &color,
    sizeof(speed), &speedInfo);
88                          SQLBindCol(subExecStat, 2, SQL_CHAR, &printType,
    sizeof(printType), &printTypeInfo);
89                          SQLBindCol(subExecStat, 3, SQL_FLOAT, &price,
    sizeof(price), &priceInfo);
90
91                          while(SQLFetch(subExecStat) != SQL_NO_DATA) {
92                              printf("model=%d, color=%s, price=%.2f, maker
    =%c, type=%s",
93                              model, color ? "true" : "false", price, maker
    , type);
94                          }
95                      }
96                  }
97              }
98      }
99
```

d)

e)
```
 1      #include <sqlcli.h>
 2      #include <string.h>
 3      ...
 4      void insertNewPC() {
 5
 6          int model, ram, hd;
 7          float speed, price;
 8          char maker;
 9
10          SQLINTEGER modelCount;
11
12          SQLHENV myEnv;
13          SQLHDBC myCon;
14          SQLHSTMT execStat, subExecStat;
15
16          SQLRETURN errorCode1, errorCode2, errorCode3;
17
18          errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
19                      SQL_NULL_HANDLE, &myEnv);
20
21          if (!errorCode1) {
22              errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon
    );
```

```
23            }
24
25        if (!errorCode2) {
26            errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
   execStat);
27        }
28
29        if (!errorCode3) {
30            printf("Enter manufacturer:\n");
31            scanf("%c", &maker);
32
33            printf("Enter model:\n");
34            scanf("%d", &model);
35
36            printf("Enter speed:\n");
37            scanf("%f", &speed);
38
39            printf("Enter ram:\n");
40            scanf("%d", &ram);
41
42            printf("Enter hd:\n");
43            scanf("%d", &hd);
44
45            printf("Enter price:\n");
46            scanf("%f", &price);
47
48            printf("Enter maker:\n");
49            scanf("%c", &maker);
50
51            SQLPrepare(execStat,
52                    "SELECT COUNT(model) FROM ("
53                    "(SELECT model FROM Product WHERE model=:model)
    "
54                    "UNION "
55                    "(SELECT model FROM PC WHERE model= ?)",
   SQL_NTS);
56                SQLBindParameter(execStat, 1, ..., model, ...);
57            SQLExecute(execStat);
58            SQLBindCol(execStat, 1, SQL_INT, &modelCount, sizeof(
   modelCount), &modelCountInfo);
59
60            if (modelCount != 0) {
61                printf("Error. Model already exists in database.");
62            } else {
63                SQLPrepare(execStat,
64                        "INSERT INTO PC(model, speed, ram, hd, price)
   "
65                        "VALUES(?, ?, ?, ?, ?)", SQL_NTS);
66                    SQLBindParameter(execStat, 1, ..., model, ...);
67                    SQLBindParameter(execStat, 2, ..., speed, ...);
68                    SQLBindParameter(execStat, 3, ..., ram, ...);
69                    SQLBindParameter(execStat, 4, ..., hd, ...);
70                    SQLBindParameter(execStat, 5, ..., price, ...);
71                SQLExecute(execStat);
```

```
72
73                SQLPrepare(execStat,
74                        "INSERT INTO Product(model, maker, type)"
75                        "VALUES(?, ?, 'pc')", SQL_NTS);
76                    SQLBindParameter(execStat, 1, ..., model, ...);
77                    SQLBindParameter(execStat, 2, ..., maker, ...);
78                SQLExecute(execStat);
79            }
80        }
81    }
82
```

2. a)
```
1    void classWithLargestPower() {
2
3        SQLINTEGER classInfo;
4        SQLCHAR class[100];
5
6        SQLHENV myEnv;
7        SQLHDBC myCon;
8        SQLHSTMT execStat, subExecStat;
9
10        SQLRETURN errorCode1, errorCode2, errorCode3;
11
12        errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
13                    SQL_NULL_HANDLE, &myEnv);
14
15        if (!errorCode1) {
16            errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon
    );
17        }
18
19        if (!errorCode2) {
20            errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    execStat);
21        }
22
23        if (!errorCode3) {
24            SQLPrepare(execStat,
25                        "SELECT class FROM FROM Classes"
26                        "WHERE numGuns * POWER(bore, 3) >= ALL ( "
27                        "SELECT numGuns * POWER(bore, 3) FROM Classes "
28                        ")", SQL_NTS);
29                SQLBindParameter(execStat, 1, ..., model, ...);
30            SQLExecute(execStat);
31            SQLBindCol(execStat, 1, SQL_CHAR, &class, sizeof(class),
    &classInfo);
32
33            while(SQLFetch(execStat) != SQL_NO_DATA) {
34                printf("Class = %s\n", class);
35            }
36        }
37    }
38
```