CSC343 Worksheet 8 Solution

June 25, 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);
```

```
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;
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]);
    }
```

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:

```
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",
```

```
                        model, speed, ram, hd, screen, price, maker);
                }
            }
        }
    }

```

c)

```
    #include <stdbool.h>
    #include <string.h>
    ...
    void printSpecifications() {
        char targetMaker;

        SQLHENV myEnv;
        SQLHDBC myCon;
        SQLHSTMT execStat, subExecStat;

        SQLINTEGER model, modelInfo, speedInfo, ram, ramInfo, hd,
    hdInfo, priceInfo, makerInfo, screen, screenInfo, color, colorInfo
    , printTypeInfo;
        SQLREAL speed, price;
        SQLCHAR maker, printType[50];

        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);
            errorCode4 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    subExecStat);
        }

        if (!errorCode3 && !errorCode4) {
            printf("Enter manufacturer:");
            scanf("%c", &targetMaker);

            SQLBindCol(execStat, 1, SQL_CHAR, &maker, sizeof(maker),
    &makerInfo);
            SQLBindCol(execStat, 2, SQL_CHAR, &productType, sizeof(
    productType), &productTypeInfo);

            while (SQLFetch(execStat) != SQL_NO_DATA) {
                if (strcmp(productType,'pc')) {
                    SQLPrepare(subExecStat,
                                "SELECT speed, ram, hd, price FROM PC
     "
```

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

```
                                    "WHERE type= ?", SQL_NTS);
                        SQLBindParameter(subExecStat, 1, ...,
productType, ...);
                    SQLExecute(subExecStat);

                    SQLBindCol(subExecStat, 1, SQL_INTEGER, &color,
sizeof(speed), &speedInfo);
                    SQLBindCol(subExecStat, 2, SQL_CHAR, &printType,
sizeof(printType), &printTypeInfo);
                    SQLBindCol(subExecStat, 3, SQL_FLOAT, &price,
sizeof(price), &priceInfo);

                    while(SQLFetch(subExecStat) != SQL_NO_DATA) {
                        printf("model=%d, color=%s, price=%.2f, maker
=%c, type=%s",
                        model, color ? "true" : "false", price, maker
, type);
                    }
                }
            }
        }
    }
```

d)

e)

```
#include <sqlcli.h>
#include <string.h>
...
void insertNewPC() {

    int model, ram, hd;
    float speed, price;
    char maker;

    SQLINTEGER modelCount;

    SQLHENV myEnv;
    SQLHDBC myCon;
    SQLHSTMT execStat, subExecStat;

    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);
    }
```

```
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, ...);
```

```
                SQLExecute(execStat);
            }
        }
    }

```

2. a)

```
    void classWithLargestPower() {

        SQLINTEGER classInfo;
        SQLCHAR class[100];

        SQLHENV myEnv;
        SQLHDBC myCon;
        SQLHSTMT execStat, subExecStat;

        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);
        }

        if (!errorCode3) {
            SQLPrepare(execStat,
                       "SELECT class FROM FROM Classes"
                       "WHERE numGuns * POWER(bore, 3) >= ALL ( "
                       "SELECT numGuns * POWER(bore, 3) FROM Classes "
                       ")", SQL_NTS);
                SQLBindParameter(execStat, 1, ..., model, ...);
            SQLExecute(execStat);
            SQLBindCol(execStat, 1, SQL_CHAR, &class, sizeof(class),
    &classInfo);

            while(SQLFetch(execStat) != SQL_NO_DATA) {
                printf("Class = %s\n", class);
            }
        }
    }

```

b)

```
    #include <sqlcli.h>
    #include <string.h>
    ...
    void countryWithMostShipsSunk() {
        char targetBattle[255];
```

```
6         char mostSunkCountry[100];
7         int maxSunkCount = 0, loopIndex = 0;
8
9         char mostDamagedCountry[100];
10        int maxDamagedCount = 0;
11
12        SQLCHAR country[100];
13        SQLINTEGER count, countInfo. countryInfo;
14
15        SQLHENV myEnv;
16        SQLHDBC myCon;
17        SQLHSTMT execStat, subExecStat;
18
19        SQLRETURN errorCode1, errorCode2, errorCode3;
20
21        errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
22                    SQL_NULL_HANDLE, &myEnv);
23
24        if (!errorCode1) {
25            errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon
    );
26        }
27
28        if (!errorCode2) {
29            errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    execStat)
30        }
31
32        if (!errorCode3) {
33
34            printf("Enter name of battle:\n");
35            scanf("%s", &targetBattle);
36
37            SQLPrepare(execStat,
38                        "SELECT country, COUNT(Outcomes.result) FROM
    Classes "
39                        "INNER JOIN Ships ON Classes.class = Ships.
    class "
40                        "INNER JOIN Outcomes ON Ships.name = Outcomes
    .ship "
41                        "INNER JOIN Battles ON Battles.name = Outcome
    .battle "
42                        "GROUP BY country "
43                        "HAVING Battles.name=:targetBattle AND"
44                        "Outcomes.result='sunk'", SQL_NTS);
45            SQLExecute(execStat);
46            SQLBindCol(execStat, 1, SQL_CHAR, &country, sizeof(
    country), &countryInfo);
47            SQLBindCol(execStat, 2, SQL_INTEGER, &count, sizeof(count
    ), &countInfo);
48
49            while(SQLFetch(execStat) != SQL_NO_DATA) {
50                if (loopIndex == 0) {
51                    strcpy(mostSunkCountry, country);
```

```
52                  }
53
54                  if (count > maxSunkCount) {
55                      maxSunkCount = count;
56                      strcpy(mostSunkCountry, country);
57                  }
58                  loopIndex = loopIndex + 1;
59              }
60
61              printf("Country with most sunk ships: %s",
    mostSunkCountry);
62
63
64              count = 0;
65              loopIndex = 0;
66              SQLPrepare(execStat,
67                          "SELECT country, COUNT(Outcomes.result) FROM
    Classes "
68                          "INNER JOIN Ships ON Classes.class = Ships.
    class "
69                          "INNER JOIN Outcomes ON Ships.name = Outcomes
    .ship "
70                          "INNER JOIN Battles ON Battles.name = Outcome
    .battle "
71                          "GROUP BY country "
72                          "HAVING Battles.name=:targetBattle AND"
73                          "Outcomes.result='damaged'", SQL_NTS);
74              SQLExecute(execStat);
75
76              while(SQLFetch(execStat) != SQL_NO_DATA) {
77                  if (loopIndex == 0) {
78                      strcpy(mostDamagedCountry, country);
79                  }
80
81                  if (count > maxDamagedCount) {
82                      maxDamagedCount = count;
83                      strcpy(mostDamagedCountry, country);
84                  }
85                  loopIndex = loopIndex + 1;
86              }
87
88              printf("Country with most damaged ships: %s",
    mostDamagedCountry);
89          }
90      }
91
```

c)

```
1   #include <sqlcli.h>
2
3   void insertClassAndShip() {
4
5       char class[100], type[2], country[100], shipName[100],
    dateLaunched[11];
6       int numGuns, bore, displacement;
```

```
7
8
9       SQLHENV myEnv;
10      SQLHDBC myCon;
11      SQLHSTMT execStat, subExecStat;
12
13      SQLRETURN errorCode1, errorCode2, errorCode3;
14
15      errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
16                  SQL_NULL_HANDLE, &myEnv);
17
18      if (!errorCode1) {
19          errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon
    );
20      }
21
22      if (!errorCode2) {
23          errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    execStat)
24      }
25
26      if (!errorCode3) {
27          printf("Enter name of class:\n");
28          scanf("%s", class);
29
30          printf("Enter name of type ('bb' or 'bc'):\n");
31          scanf("%s", type);
32
33          printf("Enter name of country:\n");
34          scanf("%s", country);
35
36          printf("Enter name of numGuns:\n");
37          scanf("%d", &numGuns);
38
39          printf("Enter name of bore:\n");
40          scanf("%d", &bore);
41
42          printf("Enter name of displacement:\n");
43          scanf("%d", &displacement);
44
45          printf("Enter name of ship (if first ship, skip by
    pressing ENTER):\n");
46          fgets(shipName, sizeof shipName,stdin);
47
48          if (shipName[0] == '\n') {
49              strncpy(shipName, class, sizeof(class));
50          }
51
52          printf("Enter date launched (YYYY-MM-DD):\n");
53          scanf("%s", dateLaunched);
54
55          SQLPrepare(execStat,
56                  "INSERT INTO Classes(class, type, country,
    numGuns, bore, displacement)"
```

```
                    "VALUES (?, ?, ?, ?, ?, ?)", SQL_NTS);
                SQLBindParameter(execStat, 1, ..., class, ...);
                SQLBindParameter(execStat, 2, ..., type, ...);
                SQLBindParameter(execStat, 3, ..., country, ...);
                SQLBindParameter(execStat, 4, ..., numGuns, ...);
                SQLBindParameter(execStat, 5, ..., bore, ...);
                SQLBindParameter(execStat, 6, ..., displacement, ...)
    ;
            SQLExecute(execStat);

            SQLPrepare(execStat,
                    "INSERT INTO Ships(name, class, launched)"
                    "VALUES (?, ?, ?)", SQL_NTS);
                SQLBindParameter(execStat, 1, ..., shipName, ...);
                SQLBindParameter(execStat, 2, ..., class, ...);
                SQLBindParameter(execStat, 3, ..., dateLaunched, ...)
    ;
            SQLExecute(execStat);
        }
    }
```

d)

```
#include <sqlcli.h>

void correctError() {
    SQLCHAR battle[101], shipName[101];
    DATE_STRUCT dateLaunched;
    DATE_STRUCT dateBattle;

    int newDateLaunchedDay;
    int newDateLaunchedMonth;
    int newDateLaunchedYear;

    int newDateBattleDay;
    int newDateBattleMonth;
    int newDateBattleYear;

    SQLCHAR name[101], class[101]

    SQLHENV myEnv;
    SQLHDBC myCon;
    SQLHSTMT execStat, subExecStat;

    SQLRETURN errorCode1, errorCode2, errorCode3, errorCode4;

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

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

    if (!errorCode2) {
```

```
32          errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    execStat);
33          errorCode4 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &
    subExecStat);
34      }
35
36      if (!errorCode3 && !errorCode4) {
37
38          SQLPrepare(execStat,
39                      "SELECT Ships.name, Ships.class, Outcomes.
    battle"
40                      "FROM Ships "
41                      "INNER JOIN Outcomes ON Ships.name = Outcomes
    .ship "
42                      "INNER JOIN Battles ON Outcomes.battle =
    Battles.name "
43                      "WHERE Ships.launched > Battles.date",
    SQL_NTS);
44          SQLExecute(execStat);
45          SQLBindCol(execStat, 1, SQL_CHAR, &shipName, sizeof(
    shipName), &shipNameInfo);
46          SQLBindCol(execStat, 2, SQL_FLOAT, &class, sizeof(class),
     &classInfo);
47          SQLBindCol(execStat, 3, SQL_INTEGER, &battle, sizeof(
    battle), &battleInfo);
48
49          while(SQLFetch(execStat) != SQL_NO_DATA) {
50
51              printf("Error. Ship %s is launched after date of
    battle.\n");
52
53              printf("Enter correct launched date (YYYY-MM-DD. Type
     0-0-0 to skip):\n");
54              scanf("%d-%d-%d", &newDateLaunchedDay, &
    newDateLaunchedMonth, &newDateLaunchedYear);
55
56              if (!newDateLaunchedDay == 0 &&
57                  !newDateLaunchedMonth == 0 &&
58                  !newDateLaunchedYear == 0) {
59                  // Correct date of launch
60                  SQLPrepare(subExecStat,
61                              "UPDATE Ships "
62                              "SET launched = ? "
63                              "WHERE name = ? AND class = ?",
    SQL_NTS);
64                      SQLBindParameter(subExecStat, 1, ...,
    newDateLaunched, ...);
65                      SQLBindParameter(subExecStat, 2, ...,
    shipName, ...);
66                      SQLBindParameter(subExecStat, 3, ..., class,
    ...);
67                  SQLExecute(subExecStat);
68              }
69
```

```
70                printf("Enter correct launched date (YYYY-MM-DD. Type
      0-0-0 to skip):\n");
71                scanf("%d-%d-%d", &newDateBattleDay, &
   newDateBattleMonth, &newDateBattleYear);
72
73                if (!newDateBattleDay == 0 &&
74                    !newDateBattleMonth == 0 &&
75                    !newDateBattleYear == 0) {
76                    // Correct date of battle
77                    SQLPrepare(subExecStat,
78                               "UPDATE Battles "
79                               "SET date = ? "
80                               "WHERE name = ?", SQL_NTS);
81                        SQLBindParameter(subExecStat, 1, ...,
   newDateBattle, ...);
82                        SQLBindParameter(subExecStat, 2, ..., battle,
      ...);
83                    SQLExecute(subExecStat);
84                }
85            }
86        }
87    }
88
```

3. a)

```
1   import java.sql.*;
2
3   class Playground {
4       public static void main(String[ ] args) {
5
6           int model, ram, hd, modelSol;
7           float targetPrice, minDiff, speedSol, minDiff = Float.
   MAX_VALUE;
8
9           String makerSol;
10
11          try {
12              // The newInstance() call is a work around for some
13              // broken Java implementations
14
15              Class.forName("com.mysql.jdbc.Driver").newInstance();
16              Connection myCon = DriverManager.getConnection("jdbc:
   mysql://localhost/Q3");
17
18              SQLPrepare(execStat,
19                      "SELECT model, speed, ram, hd, price, maker "
20                      "FROM Product NATURAL JOIN PC", SQL_NTS);
21              SQLExecute(execStat);
22              SQLBindCol(execStat, 1, SQL_INTEGER, &model, sizeof(
   model), &modelInfo);
23              SQLBindCol(execStat, 2, SQL_FLOAT, &speed, sizeof(
   speed), &speedInfo);
24              SQLBindCol(execStat, 3, SQL_INTEGER, &ram, sizeof(ram
   ), &ramInfo);
```

```
25                SQLBindCol(execStat, 4, SQL_INTEGER, &hd, sizeof(hd),
   &hdInfo);
26                SQLBindCol(execStat, 5, SQL_FLOAT, &price, sizeof(
   price), &priceInfo);
27                SQLBindCol(execStat, 6, SQL_CHAR, &maker, sizeof(
   maker), &makerInfo);
28
29                printf("Enter target price:");
30                scanf("%f", &targetPrice);
31
32                while (SQLFetch(execStat) != SQL_NO_DATA) {
33
34                    if (abs(price - targetPrice) >= minDiff) {
35                        continue;
36                    }
37
38                    minDiff = abs(price - targetPrice);
39                    modelSol = model;
40                    speedSol = speed;
41                    makerSol = maker;
42                }
43
44                printf("maker=%c, model=%d, speed=%.2f\n", makerSol,
   modelSol, speedSol);
45
46            }
47
48
49            } catch (SQLException ex) {
50                // handle the error
51                System.out.println("Error occured while establishing
   database connection");
52            }
53        }
54    }
55
```

Notes:

- JDBC
  - Setup
    1. Import JDBC

       import java.sql.*;

    2. Load a drive rfor the database system to use (i.e. sqlite, postgresql, mysql)

       Class.forName(< driver name >);

       Example:

```
Class.forName("com.mysql.jdbc.Driver");
```

3. Establish a connection to the database

   Connection myCon = DriverManager.getConnection(< URL >, < user name >, < password >);

   * The url for jdbc is

     jdbc:mysql://< host name >/< database name >

– Creating Statements in JDBC

1. **createStatement():**
   * Prepares a statement object
   * Is similar to **SQLAllocHandle** in C
2. **prepareStatement(Q):**
   * Prepares a statement object but with query
   * Is similar to **SQLAllocHandle** and **SQLPrepare** combined in C
3. **executeQuery(Q):**
   * takes a query statement Q and executes it.
   * Used for SELECT
   * Is similar to **SQLPrepare** and **SQLExecute**
4. **executeQuery():**
   * Is used with **prepareStatement(Q)**
   * Used for SELECT
   * Is similar to **SQLExecute** in C
5. **execUpdate(U):**
   * takes a non-query statement U and executes it.
   * Used for UPDATE AND INSERT
   * Is similar to **SQLPrepare** and **SQLExecute**
6. **executeQuery():**
   * Is used with **prepareStatement(Q)**
   * Used for UPDATE AND INSERT
   * Is similar to **SQLExecute** in C

Example:

```
// Example 1
Statement execStat = myCon.createStatement();
ResultSet worths = execStat.executeQuery(
    "SELECT netWorth FROM MovieExec");
```

```
    // Example 2
    PreparedStatement execStat = myCon.prepareStatement(
        "SELECT netWorth FROM MovieExec");
    ResultSet worths = execStat.executeQuery()

    // Example 3
    Statement starStat = myCon.createStatement();
    starStat.executeUpdate("INSERT INTO StarsIn VALUES('
   Remember the Titans', 2000, 'Denzel Washington')");

    // Example 4
    PreparedStatement starStat = myCon.prepareStatement(
        "INSERT INTO StarsIn VALUES('Remember the Titans',
   2000, 'Denzel Washington')");
    starStat.executeUpdate()");

```

- Cursor Operations in JDBC
  - **next():**
    - Moves to next tuple
  - **getString(i), getInt(i), getFloat(i):**
    - Fetches and converts value at ith column in tuple

    Example:

```
    Statement execStat = myCon.createStatement();
    ResultSet worths = execStat.executeQuery(
        "SELECT netWorth FROM MovieExec");

    while (worths.next()) {
        int worth = worths.getInt(1);
        ...
    }

```

- Parameter Passing
  - **Syntax:** < PreparedStatement object >.setString(< column position >, < insert variable >);
  - Is similar to **SQLBindCol** in C

    Example:

```
    PreparedStatement studioStat = myCon.prepareStatement(
        "INSERT INTO Studio(name, address) VALUES(?, ?)");

    studioStat.setString(1, studioName);
    studioStat.setString(2, studioAddr);
    studioStat.executeUpdate();

```