CSC343 Worksheet 7 Solution

June 23, 2020

1. a)

```
void askUserForPrice() {
    EXEC SQL BEGIN DECLARE SECTION;
        int model;
        float speed;
        int ram;
        int hd;
        float price;
        char maker;
        float targetPrice;

        float minDiff;
        int modelSol;
        float speedSol;
        char makerSol;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE execCursor CURSOR FOR
        SELECT * FROM Product NATURAL JOIN PC

    EXEC SQL OPEN execCursor;

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

    while(1) {
        EXEC SQL FETCH FROM execCursor INTO :model,
            :speed, :ram, :hd, :price, :maker;

        if (NO_MORE_TUPLES) break;

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

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

```
40
41          EXEC SQL CLOSE execCursor;
42
43          printf("maker=%c, model=%d, speed=%.2f\n", makerSol, modelSol
    , speedSol);
44
45      }
46
```

Notes:

- EXEC SQL
  - Allows to use SQL statements within a host-language program
- The DECLARE Section
  - is used to declare shared variables
  - **Syntax:**
    EXEC SQL BEGIN DECLARE SECTION;
    ... // Variable declarations in any language
    EXEC SQL END DECLARE SECTION;

    Example:

```
1           void getStudio() {
2               EXEC SQL BEGIN DECLARE SECTION;
3                   char studioName[50], studioAddr[256]; // <- c
    variables
4                   char SQLSTATE[6];
5               EXEC SQL END DECLARE SECTION;
6
7               EXEC SQL INSERT INTO Studio(name, address)
8                       VALUES (:studioName, :studioAddr);
9           }
10
```

- Cursors
  - Is the most versatile way to connect SQL queries
  - **Syntax:**
    EXEC SQL DECLARE < cursor name > CURSOR FOR < query >

    EXEC SQL OPEN < cursor name >;
    ...
    EXEC SQL CLOSE < cursor name >;

    Example:

```
        void getStudio() {
            EXEC SQL BEGIN DECLARE SECTION;
                char studioName[50], studioAddr[256]; // <- c variables
                char SQLSTATE[6];
            EXEC SQL END DECLARE SECTION;

            EXEC SQL INSERT INTO Studio(name, address)
                        VALUES (:studioName, :studioAddr);
        }

```

Example in Python:

```
        import sqlite3
        connection = sqlite3.connect("company.db")

        cursor = connection.cursor()

        staff_data = [ ("William", "Shakespeare", "m", "1961-10-25"),
                       ("Frank", "Schiller", "m", "1955-08-17"),
                       ("Jane", "Wall", "f", "1989-03-14") ]

        for p in staff_data:
            format_str = """INSERT INTO employee (staff_number, fname, lname, gender, birth_date)
            VALUES (NULL, "{first}", "{last}", "{gender}", "{birthdate}");"""

            sql_command = format_str.format(first=p[0], last=p[1], gender=p[2], birthdate = p[3])
            cursor.execute(sql_command)

```

- Fetch Statement
  - fetch data from the result table one row at a time
  - **Syntax:**
    EXEC SQL FETCH FROM < cursor name > INTO < list of variables >

    Example:

```
        void worthRanges() {
            int i, digits, counts[15];
            EXEC SQL BEGIN DECLARE SECTION;
                int worth;
                char SQLSTATE[6];
            EXEC SQL END DECLARE SECTION;
            EXEC SQL DECLARE execCursor CURSOR FOR
                SELECT netWorth FROM MovieExec;
```

```
9
10              EXEC SQL OPEN execCursor;
11              for (i=1; i < 15; i++) counts[i] = 0;
12              while(1) {
13                  EXEC SQL FETCH FROM execCursor INTO :worth; //
      fetches a row of value from movieExec and stores in worth
14                  if (NO_MORE_TUPLES) break;
15
16                  ...
17              }
18          }
19
```

b)

```
1   void findLaptops() {
2       EXEC SQL BEGIN DECLARE SECTION;
3           int model;
4           float speed;
5           int ram;
6           int hd;
7           int screen;
8           float price;
9
10          float minSpeed;
11          int minRam;
12          int minHd;
13          float minPrice;
14      EXEC SQL END DECLARE SECTION;
15
16      EXEC SQL DECLARE execCursor CURSOR FOR
17          SELECT model, speed, ram, hd, screen, price, maker
18          FROM Product NATURAL JOIN Laptop;
19
20      EXEC SQL OPEN execCursor;
21
22      printf("Enter minimum speed:");
23      scanf("%f", &minSpeed);
24
25      printf("Enter minimum ram:");
26      scanf("%f", &minRam);
27
28      printf("Enter minimum hard-drive space:");
29      scanf("%f", &minHd);
30
31      printf("Enter minimum price:");
32      scanf("%f", &minPrice);
33
34      while(1) {
35          EXEC SQL FETCH FROM execCursor INTO :model,
36              :speed, :ram, :hd, :screen, :price, :maker;
37
38          if (NO_MORE_TUPLES) break;
39
40          if (
41              speed >= minSpeed &&
```

```
                ram >= minRam &&
                hd >= minHd &&
                screen >= minScreen
            ) {
                printf("model=%d, speed=%.2f, ram=%d, hd=%d, screen=%
d, price=%.2f, maker=%c",
                    model, speed, ram, hd, screen, price, maker);
            }
        }

        EXEC SQL CLOSE execCursor;
    }

```

c)

```
#include <stdbool.h>
#include <string.h>
...
void printSpecifications() {
    EXEC SQL BEGIN DECLARE SECTION;
        int model;
        bool color;
        char printType[50];
        float price;

        float speed;
        int ram;
        int hd;
        int screen;

        char maker;
        int productModel;
        char productType[50];

        char targetMaker;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE execCursor CURSOR FOR
        SELECT DISTINCT maker, DISTINCT productType FROM Product;

    printf("Enter manufacturer:");
    scanf("%c", &targetMaker);

    EXEC SQL OPEN execCursor;
    while (1) {
        EXEC SQL FETCH FROM execCursor INTO :maker, :productType;

        if (NO_MORE_TUPLES) break;

        if (tolower(maker) != tolower(targetMaker)) continue;

        if (strcmp(productType,'pc')) {
            EXEC SQL DECLARE pcCursor CURSOR FOR
                SELECT speed, ram, hd, price FROM PC
                NATURAL JOIN Product
```

```
41                      WHERE type=productType;
42
43                  EXEC SQL OPEN pcCursor;
44                  while(1) {
45                      EXEC SQL FETCH FROM pcCursor INTO :speed,
46                          :ram, :hd, :price;
47
48                      if (NO_MORE_TUPLES) break;
49
50                      printf("model=%d, speed=%.2f, ram=%d, hd=%d,
    price=%.2f, maker=%c, type=%s",
51                      model, speed, ram, hd, screen, price, maker,
    productType);
52                  }
53                  EXEC SQL CLOSE pcCursor;
54
55              } else if (strcmp(productType, 'laptop')) {
56
57                  EXEC SQL DECLARE laptopCursor CURSOR FOR
58                      SELECT speed, ram, hd, screen, price FROM Laptop
59                      NATURAL JOIN Product
60                      WHERE type=productType;
61
62                  EXEC SQL OPEN laptopCursor;
63                  while(1) {
64                      EXEC SQL FETCH FROM laptopCursor INTO :speed,
65                          :ram, :hd, :screen, :price;
66
67                      if (NO_MORE_TUPLES) break;
68
69                      printf("model=%d, speed=%.2f, ram=%d, hd=%d,
    screen=%d, price=%.2f, maker=%c, type=%s",
70                      model, speed, ram, hd, screen, screen, price,
    maker, productType);
71                  }
72                  EXEC SQL CLOSE laptopCursor;
73
74
75              } else if (strcmp(productType, 'printer')) {
76                  EXEC SQL DECLARE printerCursor CURSOR FOR
77                      SELECT color, printType, price FROM Printer
78                      NATURAL JOIN Product
79                      WHERE type=productType;
80
81                  EXEC SQL OPEN printerCursor;
82                  while(1) {
83                      EXEC SQL FETCH FROM printerCursor INTO :color,
84                          :printType, :price;
85
86                      if (NO_MORE_TUPLES) break;
87
88                      printf("model=%d, color=%s, price=%.2f, maker=%c,
     type=%s",
89                      model, color ? "true" : "false", price, maker,
```

```
   type);
90                 }
91                 EXEC SQL CLOSE printerCursor;
92             }
93         }
94         EXEC SQL CLOSE execCursor;
95     }
96
```

Correct Solution:

```
1     #include <stdbool.h>
2     #include <string.h>
3     ...
4     void printSpecifications() {
5         EXEC SQL BEGIN DECLARE SECTION;
6             int model;
7             bool color;
8             char printType[50];
9             float price;
10
11            float speed;
12            int ram;
13            int hd;
14            int screen;
15
16            char maker;
17            int productModel;
18            char productType[50];
19
20            char targetMaker;
21        EXEC SQL END DECLARE SECTION;
22
23        EXEC SQL DECLARE execCursor CURSOR FOR
24            SELECT maker, productType FROM Product
25            GROUP BY maker, productType; // <- Correction
26
27        printf("Enter manufacturer:");
28        scanf("%c", &targetMaker);
29
30        EXEC SQL OPEN execCursor;
31        while (1) {
32            EXEC SQL FETCH FROM execCursor INTO :maker, :
    productType;
33
34            if (NO_MORE_TUPLES) break;
35
36            if (tolower(maker) != tolower(targetMaker)) continue;
37
38            if (strcmp(productType,'pc')) {
39                EXEC SQL DECLARE pcCursor CURSOR FOR
```

```
40                      SELECT speed, ram, hd, price FROM PC
41                      NATURAL JOIN Product
42                      WHERE type=productType;
43
44                  EXEC SQL OPEN pcCursor;
45                  while(1) {
46                      EXEC SQL FETCH FROM pcCursor INTO :speed,
47                          :ram, :hd, :price;
48
49                      if (NO_MORE_TUPLES) break;
50
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                  EXEC SQL CLOSE pcCursor;
55
56              } else if (strcmp(productType, 'laptop')) {
57
58                  EXEC SQL DECLARE laptopCursor CURSOR FOR
59                      SELECT speed, ram, hd, screen, price FROM
    Laptop
60                      NATURAL JOIN Product
61                      WHERE type=productType;
62
63                  EXEC SQL OPEN laptopCursor;
64                  while(1) {
65                      EXEC SQL FETCH FROM laptopCursor INTO :speed,
66                          :ram, :hd, :screen, :price;
67
68                      if (NO_MORE_TUPLES) break;
69
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                  EXEC SQL CLOSE laptopCursor;
74
75
76              } else if (strcmp(productType, 'printer')) {
77                  EXEC SQL DECLARE printerCursor CURSOR FOR
78                      SELECT color, printType, price FROM Printer
79                      NATURAL JOIN Product
80                      WHERE type=productType;
81
82                  EXEC SQL OPEN printerCursor;
83                  while(1) {
84                      EXEC SQL FETCH FROM printerCursor INTO :color,
85                          :printType, :price;
86
87                      if (NO_MORE_TUPLES) break;
```

```
88
89                      printf("model=%d, color=%s, price=%.2f, maker
    =%c, type=%s",
90                      model, color ? "true" : "false", price, maker,
     type);
91                  }
92                  EXEC SQL CLOSE printerCursor;
93              }
94          }
95          EXEC SQL CLOSE execCursor;
96      }
97
```

d)

e)

```
1   #include <stdbool.h>
2   #include <string.h>
3   ...
4   void insertNewPC() {
5       EXEC SQL BEGIN DECLARE SECTION;
6           int model;
7           float speed;
8           int ram;
9           int hd;
10          float price;
11          char maker;
12
13          int modelCount;
14      EXEC SQL END DECLARE SECTION;
15
16      printf("Enter manufacturer:\n");
17      scanf("%c", &maker);
18
19      printf("Enter model:\n");
20      scanf("%d", &model);
21
22      printf("Enter speed:\n");
23      scanf("%f", &speed);
24
25      printf("Enter ram:\n");
26      scanf("%d", &ram);
27
28      printf("Enter hd:\n");
29      scanf("%d", &hd);
30
31      printf("Enter price:\n");
32      scanf("%f", &price);
33
34      printf("Enter maker:\n");
35      scanf("%c", &maker);
36
37      EXEC SQL DECLARE execCursor CURSOR FOR
38          SELECT COUNT(model) FROM (
39              (SELECT model FROM Product WHERE model=:model)
```

```
                UNION
                (SELECT model FROM PC WHERE model=:model)
            );

        EXEC SQL OPEN execCursor;
            EXEC SQL FETCH FROM execCursor INTO :modelCount;

            if (modelCount != 0) {
                printf("Error. Model already exists in database.");
            } else {
                EXEC SQL INSERT INTO PC(model, speed, ram, hd, price)
                                VALUES(:model, :speed, :ram, :hd, :
price);

                EXEC SQL INSERT INTO Product(model, maker, type)
                                VALUES(:model, :maker, "pc")
            }


        EXEC SQL CLOSE execCursor;
    }

```

2. a)

```
    void classWithLargestPower() {
        EXEC SQL BEGIN DECLARE SECTION;
            int class;
        EXEC SQL END DECLARE SECTION;

        EXEC SQL SELECT class FROM FROM Classes
            INTO :class
            WHERE numGuns * POWER(bore, 3) >= ALL (
                SELECT numGuns * POWER(bore, 3) FROM Classes
            );

        printf("Class = %s\n", class);
    }

```

b)

```
    #include <string.h>
    ...
    void countryWithMostShipsSunk() {
        EXEC SQL BEGIN DECLARE SECTION;
            char targetBattle[255];
            char country[100];
            int count;

            char mostSunkCountry[100];
            int maxSunkCount = 0;

            char mostDamagedCountry[100];
            int maxDamagedCount = 0;

        EXEC SQL END DECLARE SECTION;
```

```
16
17          printf("Enter name of battle:\n");
18          scanf("%s", &targetBattle);
19
20          EXEC SQL DECLARE shipsSunkCursor CURSOR FOR
21              SELECT country, COUNT(Outcomes.result) FROM Classes
22              INNER JOIN Ships ON Classes.class = Ships.class
23              INNER JOIN Outcomes ON Ships.name = Outcomes.ship
24              INNER JOIN Battles ON Battles.name = Outcome.battle
25              GROUP BY country
26              HAVING Battles.name=:targetBattle;
27                     Outcomes.result='sunk';
28
29          EXEC SQL DECLARE shipsDamagedCursor CURSOR FOR
30              SELECT country, COUNT(Outcomes.result) FROM Classes
31              INNER JOIN Ships ON Classes.class = Ships.class
32              INNER JOIN Outcomes ON Ships.name = Outcomes.ship
33              INNER JOIN Battles ON Battles.name = Outcome.battle
34              GROUP BY country
35              HAVING Battles.name=:targetBattle;
36                      Outcomes.result='damaged';
37
38          EXEC SQL OPEN shipsSunkCursor;
39              while(1) {
40                  EXEC SQL FETCH FROM shipsSunkCursor INTO :country,
41                  :count;
42
43                  if (NO_MORE_TUPLES) break;
44
45                  if (count > maxSunkCount) {
46                      maxSunkCount = count;
47                      strcpy(mostSunkCountry, country);
48                  }
49              }
50
51              printf("Country with most sunk ships: %s",
    mostSunkCountry);
52
53          EXEC SQL CLOSE shipsSunkCursor;
54
55          EXEC SQL OPEN shipsDamagedCursor;
56              while(1) {
57                  EXEC SQL FETCH FROM shipsDamagedCursor INTO :country,
58                  :count;
59
60                  if (NO_MORE_TUPLES) break;
61
62                  if (count > maxDamagedCount) {
63                      maxDamagedCount = count;
64                      strcpy(mostDamagedCountry, country);
65                  }
66              }
67
68              printf("Country with most damaged ships: %s",
```

```
    mostDamagedCountry);
69
70          EXEC SQL CLOSE shipsDamagedCursor;
71
72      }
73
```

c)

```
1   #define NO_MORE_TUPLES ! (strcmp(SQLSTATE, "02000"));
2
3   void insertClassAndShip() {
4       EXEC SQL BEGIN DECLARE SECTION;
5           char class[100];
6           char type[2];
7           char country[100];
8           int numGuns;
9           int bore;
10          int displacement;
11
12          char shipName[100];
13          char dateLaunched[11];
14
15          char SQLSTATE[6];
16      EXEC SQL END DECLARE SECTION;
17
18      printf("Enter name of class:\n");
19      scanf("%s", class);
20
21      printf("Enter name of type ('bb' or 'bc'):\n");
22      scanf("%s", type);
23
24      printf("Enter name of country:\n");
25      scanf("%s", country);
26
27      printf("Enter name of numGuns:\n");
28      scanf("%d", &numGuns);
29
30      printf("Enter name of bore:\n");
31      scanf("%d", &bore);
32
33      printf("Enter name of displacement:\n");
34      scanf("%d", &displacement);
35
36      printf("Enter name of ship (if first ship, skip by pressing
    ENTER):\n");
37      fgets(shipName, sizeof shipName,stdin);
38
39      if (shipName[0] == '\n') {
40          strncpy(shipName, class, sizeof(class));
41      }
42
43      printf("Enter date launched (YYYY-MM-DD):\n");
44      scanf("%s", dateLaunched);
45
46      EXEC SQL INSERT INTO Classes(class, type, country, numGuns,
```

```
    bore, displacement)
47              VALUES (:class, :type, :country, :numGuns, :bore, :
    displacement);
48
49      EXEC SQL INSERT INTO Ships(name, class, launched)
50              VALUES (:shipName, :class, :dateLaunched);
51   }
52
```

d)

```
1    #define NO_MORE_TUPLES ! (strcmp(SQLSTATE, "02000"));
2
3    void correctError() {
4        EXEC SQL BEGIN DECLARE SECTION;
5            char battle[101];
6            char shipName[101];
7            char dateLaunched[11];
8            char newDateLaunched[11];
9
10           char dateBattle[11];
11           char newDateBattle[11];
12
13           char SQLSTATE[6];
14       EXEC SQL END DECLARE SECTION;
15
16       EXEC SQL DECLARE execCursor CURSOR FOR
17           SELECT Ships.name,
18                  Ships.class,
19                  Ships.launched,
20                  Outcomes.battle,
21                  Battles.date
22           FROM Ships
23           INNER JOIN Outcomes ON Ships.name = Outcomes.ship
24           INNER JOIN Battles ON Outcomes.battle = Battles.name
25           WHERE Ships.launched > Battles.date;
26
27       EXEC SQL OPEN execCursor;
28           while(1) {
29               EXEC SQL FETCH FROM execCursor INTO :shipName,
30                   :class, :dateLaunched, :battle, :dateBattle;
31
32               if (NO_MORE_TUPLES) break;
33
34               printf("Error. Ship %s is launched after date of
    battle.\n");
35
36               printf("Enter correct launched date (YYYY-MM-DD,
    Press enter to skip):\n");
37               fgets(dateLaunched, sizeof(dateLaunched), stdin);
38
39               if (dateLaunched[0] != '\n') {
40                   // Correct date of launch
41                   EXEC SQL UPDATE Ships
42                            SET launched = newDateLaunched
43                            WHERE name=:shipName AND
```

```
                                    class=:class AND
                                    launched=:dateLaunched;
                }

                printf("Enter correct battle date (YYYY-MM-DD, Press enter to skip):\n");
                fgets(dateBattle, sizeof(dateBattle),stdin);

                if (dateBattle[0] != '\n') {
                    // Correct date of battle
                    EXEC SQL UPDATE Battles
                            SET date = newDateBattle
                            WHERE name=:battle AND
                                  date=dateBattle;
                }
            }
        EXEC SQL CLOSE execCursor;

    }

```

e) **Notes:**

- PSM
  - Is also called **Persistent, Stored Modules**
  - Is very similar to function
    - Procedure → void function
    - Function → non-void function
  - **Syntax:**

    CREATE PROCEDURE < name > (< parameters >)
    < local declarations >
    < syntax body >;
  - **Syntax # 2:**

    CREATE FUNCTION < name > (< parameters list >) RETURNS < type >
    < local declarations >
    < syntax body >;

    Example:

```
        CREATE PROCEDURE Move(
            IN oldAddr VARCHAR(255),
            IN newAddr VARCHAR(255)
        )
        UPDATE MovieStar
        SET address = newAddr
        WHERE address = oldAddr;

```

- List of simple PSM statements
  1. Call-statement
     - **Syntax:** CALL < procedure name > (< argument list >)
     - is used to invoke procedure
     - is included in function

     Example:

```
    EXEC SQL CALL Foo(:x, 3)

```

  2. Return-statement
     - **Syntax:** RETURN < expression >
     - can only be appeared in function
     - evaluates the expression and sets the return-value of the function equivalent of the result

     Example:

```
    CREATE FUNCTION BandW(y INT, s CHAR(15)) RETURN BOOLEAN

    ...

    THEN RETURN TRUE;
    ELSE RETURN FALSE;

    END IF;

```

  3. Declarations of local variables
     - **Syntax:** DECLARE < name > < type >
     - Is used to declare a variable
     - Is not preserved by DBMS after a running of the function or procedure
     - Must precede executable statements

     Example:

```
    CREATE PROCEDURE SomeProc(IN studioName CHAR(15))

    DECLARE presNetWorth INTEGER;

```

  4. Assignment statements
     - **Syntax:** SET < variable > = < expression >
     - Is quite like assignment in other languages.

- The expression can be a query as long as it returns a signle value

Example:

```
CREATE PROCEDURE MeanVar(
    IN s CHAR(15),
    OUT mean REAL,
    OUT variance REAL
)

BEGIN
...
SET mean=0.0;
...
SET movieCount = (SELECT COUNT(name) FROM Movies);
SET mean= mean / movieCount;
END

```

5. Statement groups
   - **Syntax:** BEGIN ... END
   - Is used to envelope function body in Procedures and Functions
   - is like $function\{\}$ used in functions (i.e. javascript, c, java)

Example:

```
CREATE PROCEDURE MeanVar (
    IN s CHAR(15),
    OUT mean REAL,
    OUT variance REAL
)

BEGIN
    ...
END;

```

- IF ELSE
  - **Syntax:**

    IF < condition > THEN
    < statement list >
    ELSEIF < condition > THEN
    < statement list >
    ELSEIF
    ...
    ELSE
    < statement list >
    END IF;

Example:

```
CREATE FUNCTION BandW(y INT, s CHAR(15)) RETURNS BOOLEAN

BEGIN
IF NOT EXISTS(
        SELECT * FROM Movies WHERE year = y AND
            studioName = s)

THEN RETURN TRUE;
ELSEIF 1 <=
    (SELECT COUNT(*) FROM Movies WHERE year = y AND
        studioName = s AND genre = 'comedy')

THEN RETURN TRUE;
ELSE RETURN FALSE;
END IF;
END;

```

f) Loops in PSM

- Is used with cursor
- exists a break statement for loop (i.e.LEAVE < loop label >)
- **Syntax:**

  LOOP
  < statement list >
  END LOOP;

  Example:

```

```

g) For-loops

- **Syntax:**

  FOR < loop name > AS < cursor name > CURSOR FOR
  < query >
  DO
  < statement list >
  END FOR;