

CSC343 Worksheet 7 Solution

June 23, 2020

1. a)

```
1 void askUserForPrice() {
2     EXEC SQL BEGIN DECLARE SECTION;
3         int model;
4         float speed;
5         int ram;
6         int hd;
7         float price;
8         char maker;
9         float targetPrice;
10
11         float minDiff;
12         int modelSol;
13         float speedSol;
14         char makerSol;
15     EXEC SQL END DECLARE SECTION;
16
17     EXEC SQL DECLARE execCursor CURSOR FOR
18         SELECT * FROM Product NATURAL JOIN PC
19
20     EXEC SQL OPEN execCursor;
21
22     printf("Enter target price:");
23     scanf("%f", &targetPrice);
24
25     while(1) {
26         EXEC SQL FETCH FROM execCursor INTO :model,
27             :speed, :ram, :hd, :price, :maker;
28
29         if (NO_MORE_TUPLES) break;
30
31         if (abs(price - targetPrice) >= minDiff) {
32             continue;
33         }
34
35         minDiff = abs(price - targetPrice);
36         modelSol = model;
37         speedSol = speed;
38         makerSol = maker;
39     }
```

```

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

```

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

```

Example in Python:

```

1      import sqlite3
2      connection = sqlite3.connect("company.db")
3
4      cursor = connection.cursor()
5
6      staff_data = [ ("William", "Shakespeare", "m", "
variables
1961-10-25"),
7                      ("Frank", "Schiller", "m", "1955-08-17"
8                      ),
9                      ("Jane", "Wall", "f", "1989-03-14") ]
10
11      for p in staff_data:
12          format_str = """INSERT INTO employee (staff_number,
13                      fname, lname, gender, birth_date)
14                      VALUES (NULL, "{first}", "{last}", "{gender}", "{
15                      birthdate}");"""
16
17          sql_command = format_str.format(first=p[0], last=p
18          [1], gender=p[2], birthdate = p[3])
19          cursor.execute(sql_command)
20

```

• 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:

```

1      void worthRanges() {
2          int i, digits, counts[15];
3          EXEC SQL BEGIN DECLARE SECTION;
4              int worth;
5              char SQLSTATE[6];
6          EXEC SQL END DECLARE SECTION;
7          EXEC SQL DECLARE execCursor CURSOR FOR
8              SELECT netWorth FROM MovieExec;
9

```

```

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)

```

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

```

```

42         ram >= minRam &&
43         hd >= minHd &&
44         screen >= minScreen
45     ) {
46         printf("model=%d, speed=%.2f, ram=%d, hd=%d, screen=%d, price=%.2f, maker=%c",
47             model, speed, ram, hd, screen, price, maker);
48     }
49 }
50
51 EXEC SQL CLOSE execCursor;
52 }
53

```

```

c) #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 DISTINCT maker, DISTINCT productType FROM Product;
25
26     printf("Enter manufacturer:");
27     scanf("%c", &targetMaker);
28
29     EXEC SQL OPEN execCursor;
30     while (1) {
31         EXEC SQL FETCH FROM execCursor INTO :maker, :productType;
32
33         if (NO_MORE_TUPLES) break;
34
35         if (tolower(maker) != tolower(targetMaker)) continue;
36
37         if (strcmp(productType, 'pc')) {
38             EXEC SQL DECLARE pcCursor CURSOR FOR
39                 SELECT speed, ram, hd, price FROM PC
40                 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,
51 price=%.2f, maker=%c, type=%s",
52             model, speed, ram, hd, screen, price, maker,
53 productType);
54     }
55     EXEC SQL CLOSE pcCursor;
56
57 } else if (strcmp(productType, 'laptop')) {
58
59     EXEC SQL DECLARE laptopCursor CURSOR FOR
60     SELECT speed, ram, hd, screen, price FROM Laptop
61     NATURAL JOIN Product
62     WHERE type=productType;
63
64     EXEC SQL OPEN laptopCursor;
65     while(1) {
66         EXEC SQL FETCH FROM laptopCursor INTO :speed,
67             :ram, :hd, :screen, :price;
68
69         if (NO_MORE_TUPLES) break;
70
71         printf("model=%d, speed=%.2f, ram=%d, hd=%d,
72 screen=%d, price=%.2f, maker=%c, type=%s",
73             model, speed, ram, hd, screen, screen, price,
74 maker, productType);
75     }
76     EXEC SQL CLOSE laptopCursor;
77
78 } else if (strcmp(productType, 'printer')) {
79
80     EXEC SQL DECLARE printerCursor CURSOR FOR
81     SELECT color, printType, price FROM Printer
82     NATURAL JOIN Product
83     WHERE type=productType;
84
85     EXEC SQL OPEN printerCursor;
86     while(1) {
87         EXEC SQL FETCH FROM printerCursor INTO :color,
88             :printType, :price;
89
90         if (NO_MORE_TUPLES) break;
91
92         printf("model=%d, color=%s, price=%.2f, maker=%c,
93 type=%s",
94             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,
52 price=%.2f, maker=%c, type=%s",
53             model, speed, ram, hd, screen, price, maker,
54 productType);
55     }
56     EXEC SQL CLOSE pcCursor;
57
58     } else if (strcmp(productType, 'laptop')) {
59
60         EXEC SQL DECLARE laptopCursor CURSOR FOR
61             SELECT speed, ram, hd, screen, price FROM
62             NATURAL JOIN Product
63             WHERE type=productType;
64
65         EXEC SQL OPEN laptopCursor;
66         while(1) {
67             EXEC SQL FETCH FROM laptopCursor INTO :speed,
68                 :ram, :hd, :screen, :price;
69
70             if (NO_MORE_TUPLES) break;
71
72             printf("model=%d, speed=%.2f, ram=%d, hd=%d,
73 screen=%d, price=%.2f, maker=%c, type=%s",
74                 model, speed, ram, hd, screen, screen, price,
75 maker, productType);
76         }
77         EXEC SQL CLOSE laptopCursor;
78
79     } else if (strcmp(productType, 'printer')) {
80         EXEC SQL DECLARE printerCursor CURSOR FOR
81             SELECT color, printType, price FROM Printer
82             NATURAL JOIN Product
83             WHERE type=productType;
84
85         EXEC SQL OPEN printerCursor;
86         while(1) {
87             EXEC SQL FETCH FROM printerCursor INTO :color,
88                 :printType, :price;
89
90             if (NO_MORE_TUPLES) break;

```



```

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

```

d)

```

e) #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)

```

```

40         UNION
41         (SELECT model FROM PC WHERE model=:model)
42     );
43
44     EXEC SQL OPEN execCursor;
45     EXEC SQL FETCH FROM execCursor INTO :modelCount;
46
47     if (modelCount != 0) {
48         printf("Error. Model already exists in database.");
49     } else {
50         EXEC SQL INSERT INTO PC(model, speed, ram, hd, price)
51             VALUES(:model, :speed, :ram, :hd, :
price);
52
53         EXEC SQL INSERT INTO Product(model, maker, type)
54             VALUES(:model, :maker, "pc")
55     }
56
57
58     EXEC SQL CLOSE execCursor;
59 }
60

```

2. a)

```

    void classWithLargestPower() {
2        EXEC SQL BEGIN DECLARE SECTION;
3        int class;
4        EXEC SQL END DECLARE SECTION;
5
6        EXEC SQL SELECT class FROM FROM Classes
7            INTO :class
8            WHERE numGuns * POWER(bore, 3) >= ALL (
9                SELECT numGuns * POWER(bore, 3) FROM Classes
10            );
11
12        printf("Class = %s\n", class);
13    }
14

```

b)

```

#include <string.h>
2    ...
3    void countryWithMostShipsSunk() {
4        EXEC SQL BEGIN DECLARE SECTION;
5        char targetBattle[255];
6        char country[100];
7        int count;
8
9        char mostSunkCountry[100];
10       int maxSunkCount = 0;
11
12       char mostDamagedCountry[100];
13       int maxDamagedCount = 0;
14
15       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",
52     mostSunkCountry);
53
54     EXEC SQL CLOSE shipsSunkCursor;
55
56     EXEC SQL OPEN shipsDamagedCursor;
57     while(1) {
58         EXEC SQL FETCH FROM shipsDamagedCursor INTO :country,
59         :count;
60
61         if (NO_MORE_TUPLES) break;
62
63         if (count > maxDamagedCount) {
64             maxDamagedCount = count;
65             strcpy(mostDamagedCountry, country);
66         }
67     }
68
69     printf("Country with most damaged ships: %s",
```

```

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

```

```

c) #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,

```

```

47     bore, displacement)
48         VALUES (:class, :type, :country, :numGuns, :bore, :
49     displacement);
50
51     EXEC SQL INSERT INTO Ships(name, class, launched)
52         VALUES (:shipName, :class, :dateLaunched);
53 }

```

```

d) #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

```

```

44         class=:class AND
45         launched=:dateLaunched;
46     }
47
48     printf("Enter correct battle date (YYYY-MM-DD, Press
49 enter to skip):\n");
50     fgets(dateBattle, sizeof(dateBattle), stdin);
51
52     if (dateBattle[0] != '\n') {
53         // Correct date of battle
54         EXEC SQL UPDATE Battles
55             SET date = newDateBattle
56             WHERE name=:battle AND
57                 date=dateBattle;
58     }
59     EXEC SQL CLOSE execCursor;
60
61 }
62

```

3. a)

```

2  CREATE FUNCTION getNetWorth(studioName CHAR(15)) RETURN INTEGER
3
4  BEGIN
5      DECLARE presNetWorth INTEGER;
6      SET presNetWorth = (
7          SELECT netWorth FROM Studio INNER JOIN MovieExec
8          ON Studio.presC# = MovieExec.cert#
9          WHERE Studio.name = :studioName;
10
11      );
12      RETURN presNetWorth;
13  END;

```

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:

```

1      CREATE PROCEDURE Move(
2          IN oldAddr VARCHAR(255),
3          IN newAddr VARCHAR(255)
4      )
5      UPDATE MovieStar
6      SET address = newAddr
7      WHERE address = oldAddr;
8

```

• List of simple PSM statements

1. Call-statement

- **Syntax:** CALL < procedure name > (< argument list >)
- is used to invoke procedure
- is included in function

Example:

```

1      EXEC SQL CALL Foo(:x, 3)
2

```

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:

```

1      CREATE FUNCTION BandW(y INT, s CHAR(15)) RETURN BOOLEAN
2
3      ...
4
5      THEN RETURN TRUE;
6      ELSE RETURN FALSE;
7
8      END IF;
9

```

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:

```

1  CREATE PROCEDURE SomeProc(IN studioName CHAR(15))
2
3  DECLARE presNetWorth INTEGER;
4

```

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 single value

Example:

```

1  CREATE PROCEDURE MeanVar(
2      IN s CHAR(15),
3      OUT mean REAL,
4      OUT variance REAL
5  )
6
7  BEGIN
8      ...
9      SET mean=0.0;
10     ...
11     SET movieCount = (SELECT COUNT(name) FROM Movies);
12     SET mean= mean / movieCount;
13     END
14

```

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:

```

1  CREATE PROCEDURE MeanVar (
2      IN s CHAR(15),
3      OUT mean REAL,
4      OUT variance REAL
5  )
6
7  BEGIN
8      ...
9  END;
10

```


- IF ELSE

- **Syntax:**

```

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

```

Example:

```

1  CREATE FUNCTION BandW(y INT, s CHAR(15)) RETURNS BOOLEAN
2
3  BEGIN
4  IF NOT EXISTS(
5      SELECT * FROM Movies WHERE year = y AND
6      studioName = s)
7
8  THEN RETURN TRUE;
9  ELSEIF 1 <=
10     (SELECT COUNT(*) FROM Movies WHERE year = y AND
11     studioName = s AND genre = 'comedy')
12
13  THEN RETURN TRUE;
14  ELSE RETURN FALSE;
15  END IF;
16  END;
17

```

- 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:

```

1  CREATE PROCEDURE MeanVar(
2      IN s CHAR(15),
3      OUT mean REAL,
4      OUT variance REAL

```

```

5      )
6
7      BEGIN
8          ...
9          movieLoop: LOOP
10             FETCH FROM MovieCursor INTO newLength;
11             IF Not_Found THEN LEAVE movieLoop END IF;
12             SET movieCount = movieCount + 1;
13             SET mean = mean + newLength;
14             ...
15         END LOOP
16
17     END;
18

```

- For-loops

- is also used to iterate over cursor

- **Syntax:**

```

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

```

Example:

```

1      CREATE PROCEDURE MeanVar(
2          IN s CHAR(15),
3          OUT mean REAL,
4          OUT variance REAL
5      )
6
7      BEGIN
8          ...
9          FOR movieLoop AS MovieCursor CURSOR FOR
10             SELECT length FROM Movies WHERE studioName = s;
11          DO
12             FETCH FROM MovieCursor INTO newLength;
13             IF Not_Found THEN LEAVE movieLoop END IF;
14             SET movieCount = movieCount + 1;
15             SET mean = mean + length;
16             SET variance = mean + length * length;
17             ...
18         END LOOP
19
20     END;
21

```

- Exceptions in PSM

- **Syntax:**

DECLARE < where to go next > HANDLER FOR < condition list >
< statement >

– Choices for where to go

1. CONTINUE

* Continues to execute code as is

2. EXIT

* Leaves BEGIN ... END block

3. UNDO

* In addition to EXIT, any changes made to database is undone

Example:

```

1  CREATE FUNCTION GetYear(t VARCHAR(255)) RETURN INTEGER
2
3  DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
4  DECLARE Too_Many CONDITION FOR SQLSTATE '21000';
5
6  BEGIN
7      DECLARE EXIT HANDLER FOR Not_Found, Too_Many
8          RETURN NULL;
9      RETURN (SELECT year FROM Movies WHERE title = t);
10 END;
11

```

• Using PSM Functions and Procedures

- Always include CALL before Procedures
- Can be used in WHERE

Example:

```

1  INSERT INTO StarsIn(movieTitle, movieYear, starName)
2  VALUES ('Remember the Titans', GetYear('Remember the Titans
3  '),
4          'Denzel Washington');

```

b)

```

1  CREATE FUNCTION GetPersonType(name CHAR(50), address VARCHAR(255)
2  ) RETURN INTEGER
3
4  BEGIN
5      DECLARE type INTEGER;
6
7      IF NOT EXISTS (
8          SELECT * FROM MovieStar
9          WHERE name=name AND
10             address=address
11      ) AND EXISTS (
12          SELECT * FROM MovieExec

```

```

12         WHERE name=name AND
13             address=address
14     )
15
16     THEN RETURN 1;
17
18     ELSEIF EXISTS (
19         SELECT * FROM MovieStar
20         WHERE name=name AND
21             address=address
22     ) AND NOT EXISTS (
23         SELECT * FROM MovieExec
24         WHERE name=name AND
25             address=address
26     )
27
28     THEN RETURN 2;
29
30     ELSEIF EXISTS (
31         SELECT * FROM MovieStar NATURAL JOIN MovieExec
32         WHERE name=name AND
33             address=address
34     )
35
36     THEN RETURN 3;
37
38     ELSE RETURN 4;
39
40 END;
41

```

c)

```

1  CREATE PROCEDURE getTwoLongest(
2      IN studioName CHAR(50)
3  )
4
5      DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
6      DECLARE firstMovieTitle CHAR(100);
7      DECLARE secondMovieTitle CHAR(100);
8      DECLARE MovieCursor CURSOR FOR
9          SELECT * FROM Movies
10         ORDER BY length DESC
11         WHERE studioName=studioName
12         LIMIT 2;
13
14     BEGIN
15         set movieCount = 0;
16         movieLoop:LOOP
17
18         END LOOP;
19     END;
20

```