Name: Mihaela McReynolds

CITC 2340 – Lab 7: Procedures and Functions

(35 Points)

Refer to Chapter 12 of the Price text, chapters on PL/SQL in the Reference Guide, the PL/SQL library ebook and/or PL/SQL Web tutorials for more information about procedures, functions, and PL/SQL code blocks.

To earn your grade on this assignment, submit a text file containing the code for each procedure or function that also shows proof that you tested each one. Tests should show that the code runs correctly when it receives correct input as well as works properly when incorrect input is received (where applicable). For questions that have multiple parts, read all parts before writing the code for the first part.

1. a. Create and test a simple procedure named *check\_available\_credit* that accepts a customer id and a purchase amount as input. This procedure will then compare the purchase amount to the credit limit of the specified customer in the *demo\_customer* table. If the credit limit is high enough for the customer to make the purchase, the procedure will display the message “Credit approved for customer# nnnnn.” Otherwise display “Credit denied for customer# nnnnn.” Practice calling the procedure for various customers and purchase amounts. Include proof of your tests in your submitted lab document. (6 pts)

SQL> set serveroutput on

SQL> create or replace PROCEDURE check\_available\_credit(

2 p\_custid IN demo\_customer.custid%TYPE,

3 v\_purchase\_amount IN NUMBER

4 ) AS

5 p\_check\_credit INTEGER;

6 BEGIN

7 SELECT creditlimit -- implicit cursor

8 INTO p\_check\_credit

9 FROM demo\_customer

10 WHERE custid = p\_custid;

11 -- if the purchase amount is less than or eqal to the credit limit for specified customer,

12 -- then approve purchase

13 IF v\_purchase\_amount <= p\_check\_credit THEN

14 dbms\_output.put\_line ('Purchase approved for customer # ' || p\_custid);

15 else

16 -- deny the purchase if the credit limit is lower than the purchase amount

17 dbms\_output.put\_line('Purchase denied for customer ' || p\_custid);

18 END IF;

19 EXCEPTION -- handle all exceptions to show an error message to the user

20 WHEN OTHERS THEN

21 dbms\_output.put\_line('Error');

22 END check\_available\_credit;

Procedure CHECK\_AVAILABLE\_CREDIT compiled

SQL> create or replace PROCEDURE check\_available\_credit(

2 p\_custid IN demo\_customer.custid%TYPE,

3 v\_purchase\_amount IN NUMBER

4 ) AS

5 p\_check\_credit INTEGER;

6 BEGIN

7 SELECT creditlimit -- implicit cursor

8 INTO p\_check\_credit

9 FROM demo\_customer

10 WHERE custid = p\_custid;

11 -- if the purchase amount is less than or eqal to the credit limit for specified customer,

12 -- then approve purchase

13 IF v\_purchase\_amount <= p\_check\_credit THEN

14 dbms\_output.put\_line ('Credit approved for customer # ' || p\_custid);

15 else

16 -- deny the purchase if the credit limit is lower than the purchase amount

17 dbms\_output.put\_line('Credit denied for customer ' || p\_custid);

18 END IF;

19 EXCEPTION -- handle all exceptions to show an error message to the user

20 WHEN OTHERS THEN

21 dbms\_output.put\_line('Error');

22 END check\_available\_credit;

Procedure CHECK\_AVAILABLE\_CREDIT compiled

--TEST

-- customer 100 has a credit limit of 5000, this should be approved

SQL> call check\_available\_credit(p\_custid=>100,v\_purchase\_amount=>5000

2 );

Credit approved for customer # 100

Call completed.

-- customer 100 has a credit limit of 500, purchase should be denied for 50,000

SQL> call check\_available\_credit(p\_custid=>100,v\_purchase\_amount=>50000

2 );

Credit denied for customer 100

Call completed.

-- test with user input from the command line

SQL> call check\_available\_credit(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

2 );

old:call check\_available\_credit(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

)

new:call check\_available\_credit(p\_custid=>101,v\_purchase\_amount=>100

)

Credit approved for customer # 101

Call completed.

b. Add an exception section to the procedure that traps the NO\_DATA\_FOUND predefined exception and indicates to the user that the customer id they supplied is not valid. Also add a *when others* clause to print an appropriate message for all other errors that might occur (lumped together in one category). Save a copy of the new code as *check\_available\_credit\_v2* and test it using both valid and invalid customer ids. (2 pts)

SQL> create or replace PROCEDURE check\_available\_credit\_v2(

2 p\_custid IN demo\_customer.custid%TYPE,

3 v\_purchase\_amount IN NUMBER

4 ) AS

5 p\_check\_credit INTEGER;

6 BEGIN

7 SELECT creditlimit -- implicit cursor

8 INTO p\_check\_credit

9 FROM demo\_customer

10 WHERE custid = p\_custid;

11 -- if the purchase amount is less than or eqal to the credit limit for specified customer,

12 -- then approve purchase

13 IF v\_purchase\_amount <= p\_check\_credit THEN

14 dbms\_output.put\_line ('Credit approved for customer # ' || p\_custid);

15 else

16 -- deny the purchase if the credit limit is lower than the purchase amount

17 dbms\_output.put\_line('Credit denied for customer ' || p\_custid);

18 END IF;

19 EXCEPTION

20 -- handle the situation in which an invalid custid is entered

21 WHEN no\_data\_found then

22 dbms\_output.put\_line ('EMPLOYEE NOT FOUND');

23 -- handle all exceptions to show an error message to the user

24 WHEN OTHERS THEN

25 dbms\_output.put\_line('Unknown Error');

26 END check\_available\_credit\_v2;

Procedure CHECK\_AVAILABLE\_CREDIT\_V2 compiled

-- TEST

-- valid customer #102

SQL> call check\_available\_credit\_v2(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

2 );

old:call check\_available\_credit\_v2(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

)

new:call check\_available\_credit\_v2(p\_custid=>102,v\_purchase\_amount=>90000

)

Credit denied for customer 102

Call completed.

-- invalid customer #1000

SQL> call check\_available\_credit\_v2(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

2 );

old:call check\_available\_credit\_v2(p\_custid=>&customer\_ID,v\_purchase\_amount=>&purchase\_amount

)

new:call check\_available\_credit\_v2(p\_custid=>1000,v\_purchase\_amount=>1

)

EMPLOYEE NOT FOUND

Call completed.

2. Create and test a procedure named *check\_product\_available* that accepts a product id as input and uses the *more\_products* table to display an appropriate, informative message depending on whether or not the product is available. Be sure to correctly handle the problem of a bad product id being input and test to prove that it is working. (7 pts)

SQL> create or replace PROCEDURE check\_product\_available(

2 p\_prd\_id IN more\_products.prd\_id%TYPE

3 ) AS

4 p\_prd more\_products.prd\_id%TYPE;

5 BEGIN

6 SELECT prd\_id -- implicit cursor

7 INTO p\_prd

8 FROM more\_products

9 WHERE prd\_id = p\_prd\_id;

10 -- if the product id is in the table, display that it is available

11 dbms\_output.put\_line('Product is available');

12 EXCEPTION

13 -- handle the situation in which an invalid product id is entered

14 WHEN no\_data\_found then

15 dbms\_output.put\_line ('Product is not available');

16 -- handle all exceptions to show an error message to the user

17 WHEN OTHERS THEN

18 dbms\_output.put\_line('Unknown Error');

19 END check\_product\_available;

-- TEST

-- use available product id - 1

SQL> call check\_product\_available(p\_prd\_id=>&product\_ID

2 );

old:call check\_product\_available(p\_prd\_id=>&product\_ID

)

new:call check\_product\_available(p\_prd\_id=>1

)

Product is available

Call completed.

-- use unavailable product id - 12

SQL> call check\_product\_available(p\_prd\_id=>&product\_ID

2 );

old:call check\_product\_available(p\_prd\_id=>&product\_ID

)

new:call check\_product\_available(p\_prd\_id=>12

)

Product is not available

Call completed.

Procedure CHECK\_PRODUCT\_AVAILABLE compiled

3. Write a function named *salary\_increase* that will accept 2 input values: a specific % salary increase and an employee id. The function will return the proposed new salary for the specified employee, but will not actually change the salary. A manager might use this type of function to decide on the % increase before applying it. Be sure to handle the problem of *employee id not found*. Use your *more*\_*employees* table. (5 pts)

create or replace FUNCTION salary\_increase (

v\_salary\_increase IN NUMBER,

v\_employee\_id IN more\_employees.employee\_id%TYPE

) RETURN VARCHAR2 AS

v\_salary VARCHAR2(50);

BEGIN

SELECT salary

INTO v\_salary

FROM more\_employees

WHERE employee\_id = v\_employee\_id;

-- take the percentage and increase the salary

RETURN v\_salary + (v\_salary \* (v\_salary\_increase/100));

-- this is probably not proper exception handling, but it was the only way I could get it to display the problem to the user without using dbms\_output or showing the error name/number

EXCEPTION

WHEN no\_data\_found then

return('Please enter a valid employee ID');

END salary\_increase;

Function SALARY\_INCREASE compiled

-- TEST a valid employee ID

SQL>

SQL> set echo on;

SQL> set linesize 35

SQL> SELECT TRIM(salary\_increase(v\_salary\_increase=>&Salary\_increase\_Percent, v\_employee\_id=>&employee\_ID)) AS "New Salary"

2 from dual;

old:SELECT TRIM(salary\_increase(v\_salary\_increase=>&Salary\_increase\_Percent, v\_employee\_id=>&employee\_ID)) AS "New Salary"

from dual

new:SELECT TRIM(salary\_increase(v\_salary\_increase=>10, v\_employee\_id=>5)) AS "New Salary"

from dual

New Salary

-----------------------------------

44000

-- not a valid employee # - 100

SQL> set echo on;

SQL> set linesize 35

SQL> SELECT TRIM(salary\_increase(v\_salary\_increase=>&Salary\_increase\_Percent, v\_employee\_id=>&employee\_ID)) AS "New Salary"

2 from dual;

old:SELECT TRIM(salary\_increase(v\_salary\_increase=>&Salary\_increase\_Percent, v\_employee\_id=>&employee\_ID)) AS "New Salary"

from dual

new:SELECT TRIM(salary\_increase(v\_salary\_increase=>10, v\_employee\_id=>100)) AS "New Salary"

from dual

New Salary

-----------------------------------

Please enter a valid employee ID

4. Create a procedure named *employee\_list* that accepts a division id as an input parameter and displays the employee name, job id, and salary for each employee in that division. Use the *employees2* table. Your code block should correctly handle division ids that don’t exist in the table.

Your messages should display in the following format:

Employee *<ename>* is a <job id> for division <division id> who makes $<salary> per year.

When displaying this information, trim the extra spaces as much as possible. Test the procedure with multiple valid and invalid division identifiers. (7 pts)

SQL> create or replace PROCEDURE employee\_list(

2 p\_division\_id IN employees2.division\_id%TYPE

3 ) AS

4 p\_employees employees%ROWTYPE;

5 -- variables for the cursor

6 fname employees2.first\_name%TYPE;

7 lname employees2.last\_name%TYPE;

8 job employees2.job\_id%TYPE;

9 div employees2.division\_id%TYPE;

10 sal employees2.salary%TYPE;

11 -- use a counter to keep track of how many results I get

12 counter integer:=0;

13 -- declare cursor

14 cursor c\_employees\_div is select first\_name, last\_name, job\_id, division\_id, salary from employees2

15 where division\_id = p\_division\_id;

16 BEGIN

17 -- begin cursor

18 begin

19 open c\_employees\_div;

20 -- loop through and fetch into my variables

21 loop

22 fetch c\_employees\_div into fname, lname, job, div, sal;

23 -- exit loop after the last row of not-null values

24 exit when c\_employees\_div%NOTFOUND;

25 -- increase counter while fetching

26 counter:= counter + 1;

27 -- if the counter has increased at all, display the information

28 if (counter >0) then

29 dbms\_output.put\_line('Employee '|| fname ||' '||lname|| ' is a '||job||' for division '||div||' who makes $'||sal||' per year.');

30 end if;

31 -- end loop

32 end loop;

33 -- if, after the loop if over, the counter is still at 0, then display message

34 if (counter = 0) then

35 dbms\_output.put\_line ('Division is not available');

36 end if;

37 -- close cursor

38 close c\_employees\_div;

39 END employee\_list;

40 -- thie part doesn't work?

41 EXCEPTION

42 -- handle the situation in which an invalid division id is entered

43 WHEN no\_data\_found then

44 dbms\_output.put\_line ('Division is not available');

45 -- handle all exceptions to show an error message to the user

46 WHEN OTHERS THEN

47 dbms\_output.put\_line('Unknown Error');

48 end;

49 /

Procedure EMPLOYEE\_LIST compiled

-- TEST with valid division

SQL>

SQL> call employee\_list ('&Division');

old:call employee\_list ('&Division')

new:call employee\_list ('OPE')

Employee Kevin Black is a MGR for division OPE who makes $225000 per year.

Employee Keith Long is a MGR for division OPE who makes $165000 per year.

Employee Frank Howard is a WOR for division OPE who makes $125000 per year.

Employee Doreen Penn is a WOR for division OPE who makes $145000 per year.

Employee Megan Craig is a ENG for division OPE who makes $245000 per year.

Employee Tony Clerke is a MGR for division OPE who makes $200000 per year.

Employee Terry Cliff is a MGR for division OPE who makes $215000 per year.

Call completed.

-- TEST with invalid division

SQL>

SQL> call employee\_list ('&Division');

old:call employee\_list ('&Division')

new:call employee\_list ('ort')

Division is not available

Call completed.

5. Create a procedure named *customer\_charters* that allows a user to specify an aviation customer number as an input parameter then displays a list of the charters booked by that customer. This procedure should use the *avia\_customer* and *charter* tables from your Aircraft Charter schema. Be sure to examine the structure of these tables before attempting to write code. In this procedure, display the customer code, customer full name, charter date, charter destination, distance flown, gallons of fuel used and calculated miles per gallon of fuel used for each charter that was booked. Finally display the total miles flown for all charters that we’ve booked. Use of a cursor that retrieves the columns from the tables might be required to accomplish this task. Be sure to add error processing for invalid customer codes. Your code must correctly (non-fatally) handle the case of a customer who hasn’t yet booked a charter. Sentences that display the detail and total distance flown should have a user-friendly style.

For full credit, tests must include:

* customers who booked a single charter;
* customers who booked multiple charters;
* customers who exist but may not yet have booked a charter;
* non-existent customers

All tests must be included, even if that means adding more data to your tables. Feel free to use yourself as a customer as necessary.

(8 pts)

SQL> create or replace PROCEDURE customer\_charters (

2 p\_avia\_cust IN avia\_customer.cus\_code%TYPE

3 ) AS

4 p\_cust avia\_customer%ROWTYPE;

5 -- variables for the cursor

6 code avia\_customer.cus\_code%TYPE;

7 lname avia\_customer.cus\_lname%TYPE;

8 fname avia\_customer.cus\_fname%TYPE;

9 initial avia\_customer.cus\_initial%TYPE;

10 charter\_date charter.char\_date%TYPE;

11 charter\_dest charter.char\_destination%TYPE;

12 distance charter.char\_distance%TYPE;

13 gallons charter.char\_fuel\_gallons%TYPE;

14

15 -- declare my calculated fields

16 mpg NUMBER := 0;

17 total\_miles NUMBER := 0;

18

19 -- use a counter to keep track of how many results I get

20 counter integer:=0;

21

22 -- declare cursor

23 cursor c\_avia is

24 select avia\_customer.cus\_code, avia\_customer.cus\_lname, avia\_customer.cus\_fname,

25 avia\_customer.cus\_initial, charter.char\_date, charter.char\_destination,

26 charter.char\_distance, charter.char\_fuel\_gallons,

27 char\_distance/char\_fuel\_gallons as mpg

28 from avia\_customer LEFT OUTER JOIN charter on avia\_customer.cus\_code = charter.cus\_code

29 where p\_avia\_cust = avia\_customer.cus\_code;

30

31 BEGIN

32 -- begin cursor

33 begin

34 open c\_avia;

35 -- loop through and fetch into my variables

36 loop

37 fetch c\_avia into code, lname, fname, initial, charter\_date, charter\_dest, distance, gallons, mpg; --total\_miles

38 -- exit loop after the last row of not-null values

39 exit when c\_avia%NOTFOUND;

40 -- increase counter while fetching

41 counter:= counter + 1;

42 -- add the total distance

43 total\_miles:= total\_miles + distance;

44 -- if this customer has traveled any miles

45 if (total\_miles > 0) then

46 -- if the counter has increased at all, display the information

47 if (counter >0) then

48 dbms\_output.put\_line(fname ||' '|| initial || '. '|| lname || ' booked a charter flight on '||

49 TRIM(TO\_CHAR(TO\_DATE(charter\_date, 'DD-MON-YY'), 'Month ddSPth YYYY')) || ' to ' || charter\_dest ||

50 '. The distance was ' || distance || ' miles, and it used ' ||

51 gallons || ' gallons of gas. MPG was ' || ltrim(to\_char(mpg, '9990.00')) || '.');

52 end if;

53 else -- if no miles were traveled by this customer, then they have never booked a charter flight

54 dbms\_output.put\_line(fname ||' '|| initial || '. '|| lname || ' has never booked a charter flight.');

55 end if;

56 -- end loop

57 end loop;

58 -- if, after the loop if over, the counter is still at 0, then display message

59 if (counter = 0) then

60 dbms\_output.put\_line ('Customer code is not valid.');

61 end if;

62 -- close cursor

63 close c\_avia;

64 -- if this customer has traveled any miles, calculate their total mileage

65 if (total\_miles >0) then

66 dbms\_output.put\_line('The total mileage for customer ' || code || ' was ' || total\_miles || '.');

67 end if;

68 END customer\_charters;

69

70 -- this part doesn't work?

71 EXCEPTION

72 -- handle the situation in which an invalid division id is entered

73 WHEN no\_data\_found then

74 dbms\_output.put\_line ('Customer code is not valid.');

75 -- handle all exceptions to show an error message to the user

76 WHEN OTHERS THEN

77 dbms\_output.put\_line('Unknown Error');

78 end;

79 /

Procedure CUSTOMER\_CHARTERS compiled

SQL> -- TEST for customer who booked one charter

SQL> call customer\_charters ('&Customer\_code');

old:call customer\_charters ('&Customer\_code')

new:call customer\_charters ('10010')

Alfred A. Ramas booked a charter flight on February eighth 2013 to MOB. The distance was 884 miles, and it used 215.1 gallons of gas. MPG was 4.11.

The total mileage for customer 10010 was 884.

Call completed.

SQL> -- TEST for customer who booked multiple charters

SQL> call customer\_charters ('&Customer\_code');

old:call customer\_charters ('&Customer\_code')

new:call customer\_charters ('10011')

Leona K. Dunne booked a charter flight on February fifth 2013 to ATL. The distance was 936 miles, and it used 354.1 gallons of gas. MPG was 2.64.

Leona K. Dunne booked a charter flight on February sixth 2013 to ATL. The distance was 1023 miles, and it used 397.7 gallons of gas. MPG was 2.57.

Leona K. Dunne booked a charter flight on February eighth 2013 to TYS. The distance was 644 miles, and it used 174.3 gallons of gas. MPG was 3.69.

Leona K. Dunne booked a charter flight on February ninth 2013 to MQY. The distance was 312 miles, and it used 67.2 gallons of gas. MPG was 4.64.

The total mileage for customer 10011 was 2915.

Call completed.

SQL> -- TEST for customer who has never booked any charters

SQL> call customer\_charters ('&Customer\_code');

old:call customer\_charters ('&Customer\_code')

new:call customer\_charters ('10015')

Amy B. O'Brian has never booked a charter flight.

Call completed.

SQL> -- TEST for invalid customer

SQL> call customer\_charters ('&Customer\_code');

old:call customer\_charters ('&Customer\_code')

new:call customer\_charters ('1')

Customer code is not valid.

Call completed.