**Individual Assignment 2 – PL/SQL Block Structures – Siddharth Bahekar – 002417718**

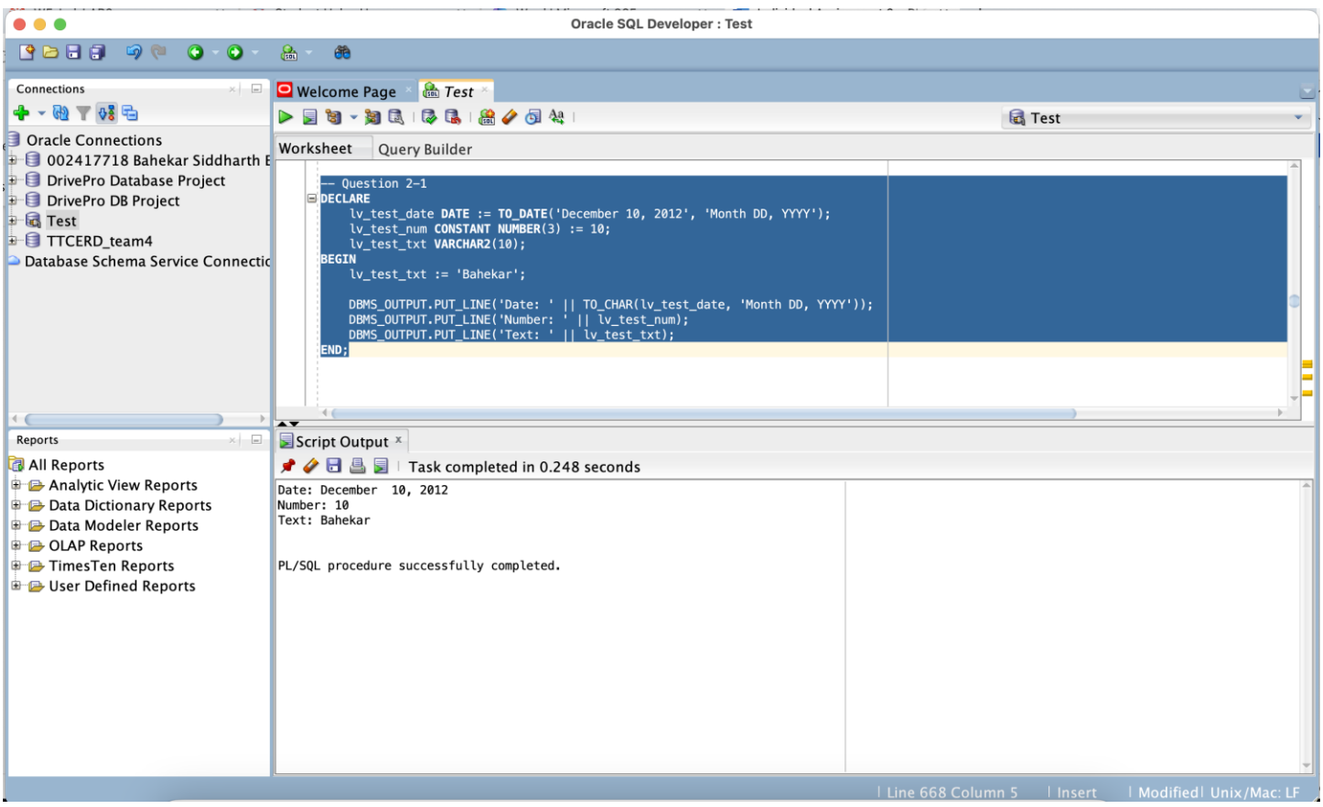
**Question 2 -1:**

**Using Scalar Variables**, create a PL/SQL block containing the following variables:

| Name | Data Type | Option | Initial Value |
|---|---|---|---|
| lv_test_date | DATE | | December 10, 2012 |
| lv_test_num | NUMBER(3) | CONSTANT | 10 |
| lv_test_txt | VARCHAR2(10) | | |

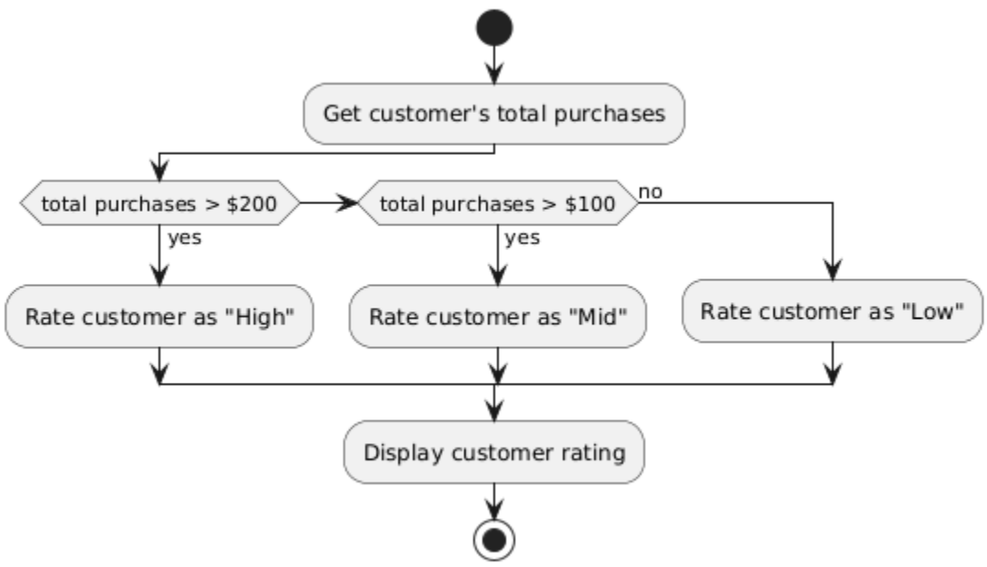
Assign your last name as the value of the text variable in the executable section of the block. Include statements in the block to display each variable's value onscreen.

```
DECLARE
   lv_test_date DATE := TO_DATE('December 10, 2012', 'Month DD, YYYY');
   lv_test_num CONSTANT NUMBER(3) := 10;
   lv_test_txt VARCHAR2(10);
BEGIN
   lv_test_txt := 'Bahekar';
   DBMS_OUTPUT.PUT_LINE('Date: ' || TO_CHAR(lv_test_date, 'Month DD, YYYY'));
   DBMS_OUTPUT.PUT_LINE('Number: ' || lv_test_num);
   DBMS_OUTPUT.PUT_LINE('Text: ' || lv_test_txt);
END;
```



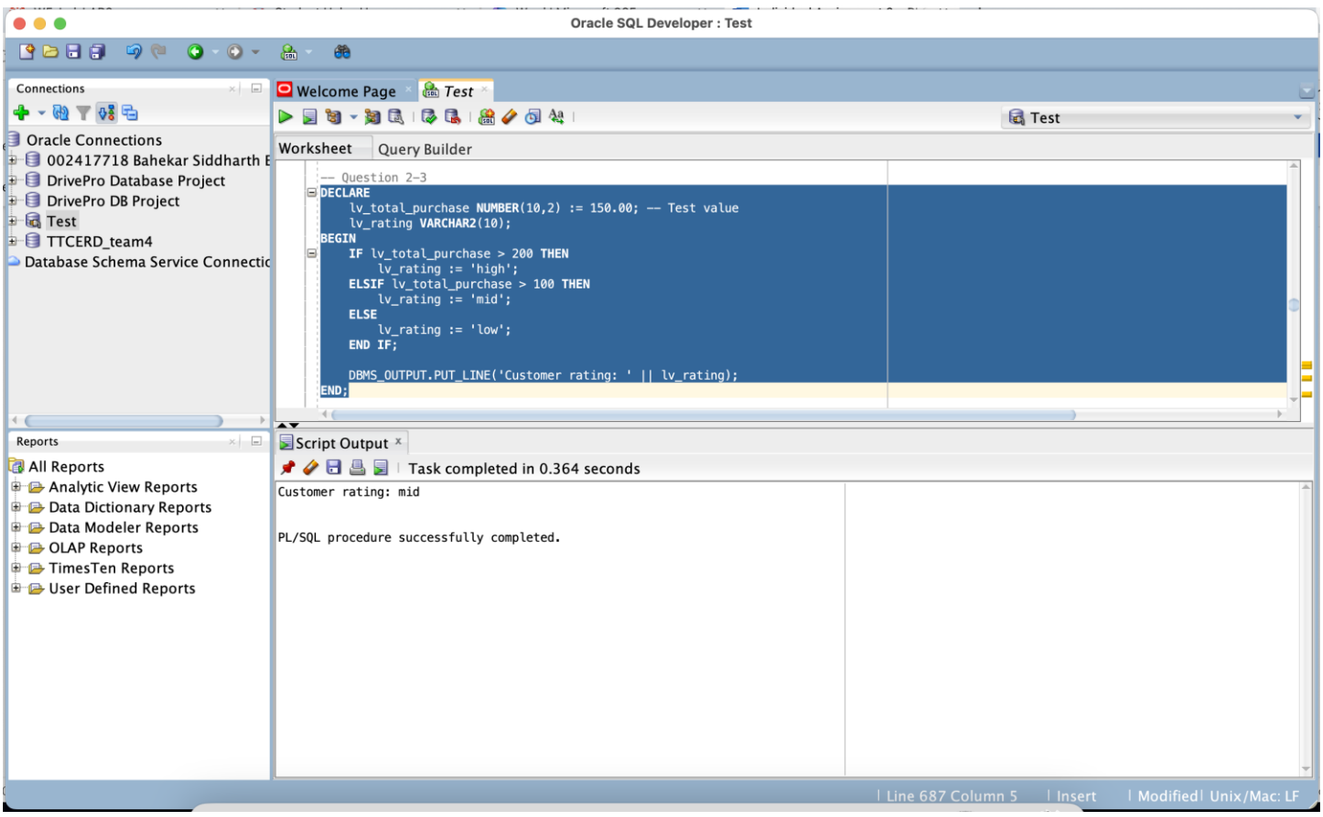
Question 2-2: Creating a Flowchart

The Brewbean’s application needs a block that determines whether a customer is rated high, mid, or low based on his or her total purchases. The block needs to determine the rating and then display the results onscreen. The code rates the customer high if total purchases are greater than $200, mid if greater than $100, and low if $100 or lower. Develop a flowchart to outline the conditional processing steps needed for this block.



Question 2-3: Using IF Statements
Create a block using an If statement to perform the actions described in Assignment 2-2. Use a scalar variable for the total purchase amount and initialize this variable to different values to test your block.

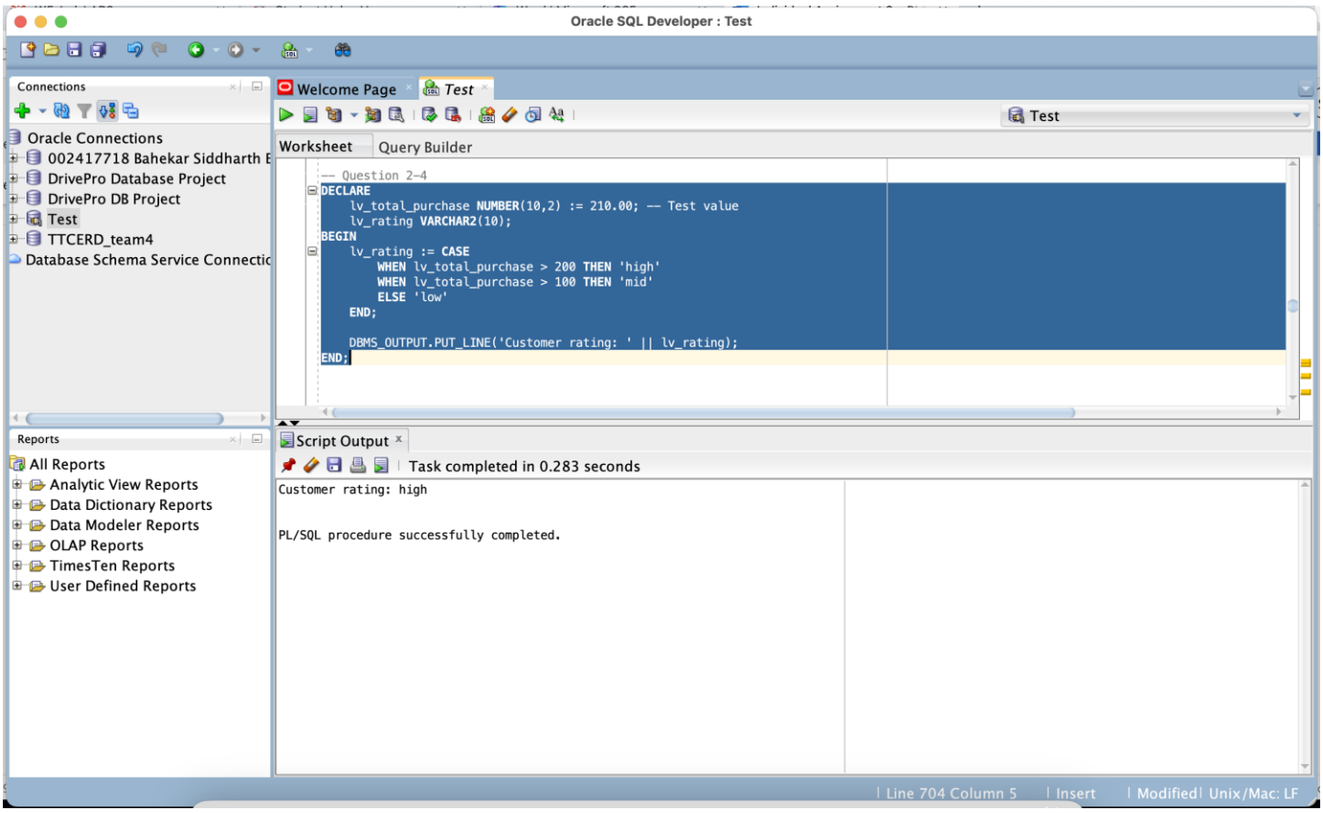
```
DECLARE
    lv_total_purchase NUMBER(10,2) := 150.00; -- Test value
    lv_rating VARCHAR2(10);
BEGIN
    IF lv_total_purchase > 200 THEN
        lv_rating := 'high';
    ELSIF lv_total_purchase > 100 THEN
        lv_rating := 'mid';
    ELSE
        lv_rating := 'low';
    END IF;
     DBMS_OUTPUT.PUT_LINE('Customer rating: ' || lv_rating);
END;
```




* Enter different values to check the rating, currently 150 has been used and it gives rating mid.

Question 2-4: Using CASE Statements
Create a block using a CASE statement to perform the actions described in Assignment 2-2. Use a scalar variable for the total purchase amount, and initialize this variable to different values.

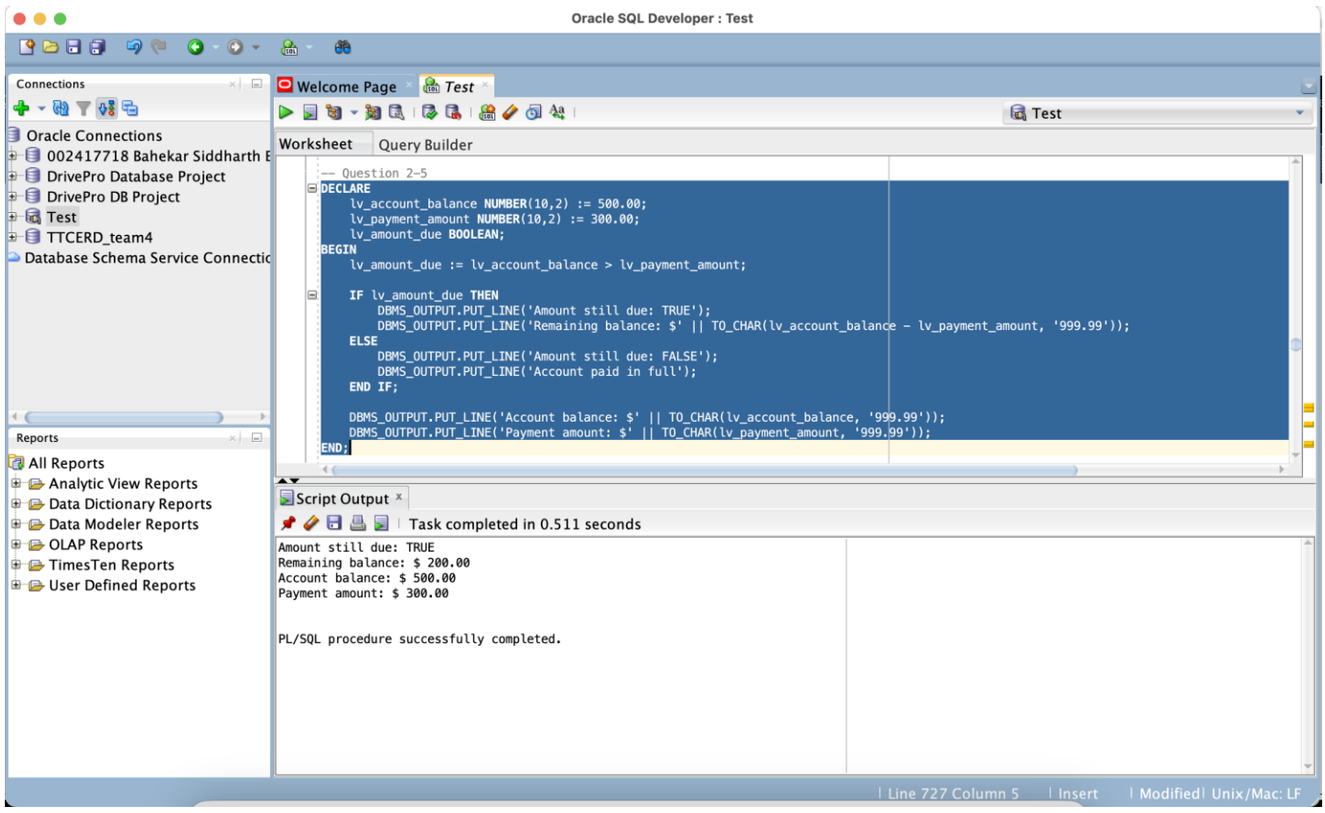
```
DECLARE
   lv_total_purchase NUMBER(10,2) := 210.00; -- Test value
   lv_rating VARCHAR2(10);
BEGIN
   lv_rating := CASE
      WHEN lv_total_purchase > 200 THEN 'high'
      WHEN lv_total_purchase > 100 THEN 'mid'
      ELSE 'low'
   END;

   DBMS_OUTPUT.PUT_LINE('Customer rating: ' || lv_rating);
END;
```




* Enter different values to check the rating, currently 210 has been used and it gives rating high.

Question 2-5: Using a Boolean Variable
Brewbean's needs program code to indicate whether an amount is still due on an account when a payment is received. Create a PL/SQL block using a Boolean variable to indicate whether an amount is still due. Declare and initialize two variables to provide input for the account balance and the payment amount received. A TRUE Boolean value should indicate an amount is still owed, and a FALSE value should indicate the account is paid in full. Use output statements to confirm that the Boolean variable is working correctly.

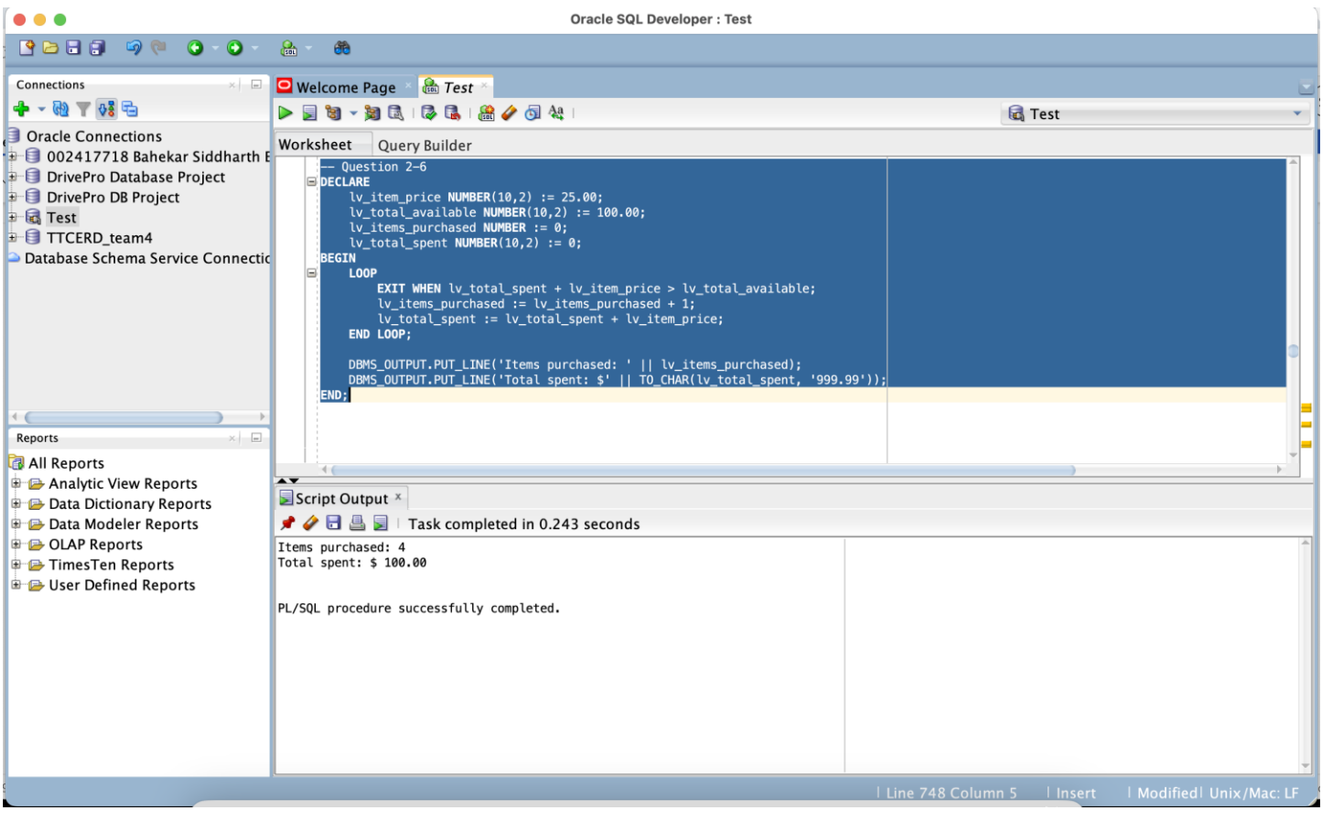
```
DECLARE
   lv_account_balance NUMBER(10,2) := 500.00;
   lv_payment_amount NUMBER(10,2) := 300.00;
   lv_amount_due BOOLEAN;
BEGIN
   lv_amount_due := lv_account_balance > lv_payment_amount;

   IF lv_amount_due THEN
      DBMS_OUTPUT.PUT_LINE('Amount still due: TRUE');
      DBMS_OUTPUT.PUT_LINE('Remaining balance: $' || TO_CHAR(lv_account_balance - lv_payment_amount, '999.99'));
   ELSE
      DBMS_OUTPUT.PUT_LINE('Amount still due: FALSE');
      DBMS_OUTPUT.PUT_LINE('Account paid in full');
   END IF;

   DBMS_OUTPUT.PUT_LINE('Account balance: $' || TO_CHAR(lv_account_balance, '999.99'));
   DBMS_OUTPUT.PUT_LINE('Payment amount: $' || TO_CHAR(lv_payment_amount, '999.99'));
END;
```


Oracle SQL Developer : Test
Connections
Oracle Connections
002417718 Bahekar Siddharth E
DrivePro Database Project
DrivePro DB Project
Test
TTCERD_team4
Database Schema Service Connectic
Reports
All Reports
Analytic View Reports
Data Dictionary Reports
Data Modeler Reports
OLAP Reports
TimesTen Reports
User Defined Reports
Welcome Page
Test
Test
Worksheet
Query Builder
-- Question 2-5
DECLARE
lv_account_balance NUMBER(10,2) := 500.00;
lv_payment_amount NUMBER(10,2) := 300.00;
lv_amount_due BOOLEAN;
BEGIN
lv_amount_due := lv_account_balance > lv_payment_amount;
IF lv_amount_due THEN
DBMS_OUTPUT.PUT_LINE('Amount still due: TRUE');
DBMS_OUTPUT.PUT_LINE('Remaining balance: $' || TO_CHAR(lv_account_balance - lv_payment_amount, '999.99'));
ELSE
DBMS_OUTPUT.PUT_LINE('Amount still due: FALSE');
DBMS_OUTPUT.PUT_LINE('Account paid in full');
END IF;
DBMS_OUTPUT.PUT_LINE('Account balance: $' || TO_CHAR(lv_account_balance, '999.99'));
DBMS_OUTPUT.PUT_LINE('Payment amount: $' || TO_CHAR(lv_payment_amount, '999.99'));
END;
Script Output
Task completed in 0.511 seconds
Amount still due: TRUE
Remaining balance: $ 200.00
Account balance: $ 500.00
Payment amount: $ 300.00
PL/SQL procedure successfully completed.
Line 727 Column 5
Insert
Modified
Unix/Mac: LF

Question 2-6: Using Looping Statements

Create a block using a loop that determines the number of items that can be purchased based on the item prices and the total available to spend. Include one initialized variable to represent the price and another to represent the total available to spend. (You could solve it with division, but you need to practice using loop structures.) The block should include statements to display the total number of items that can be purchased and the total amount spent.

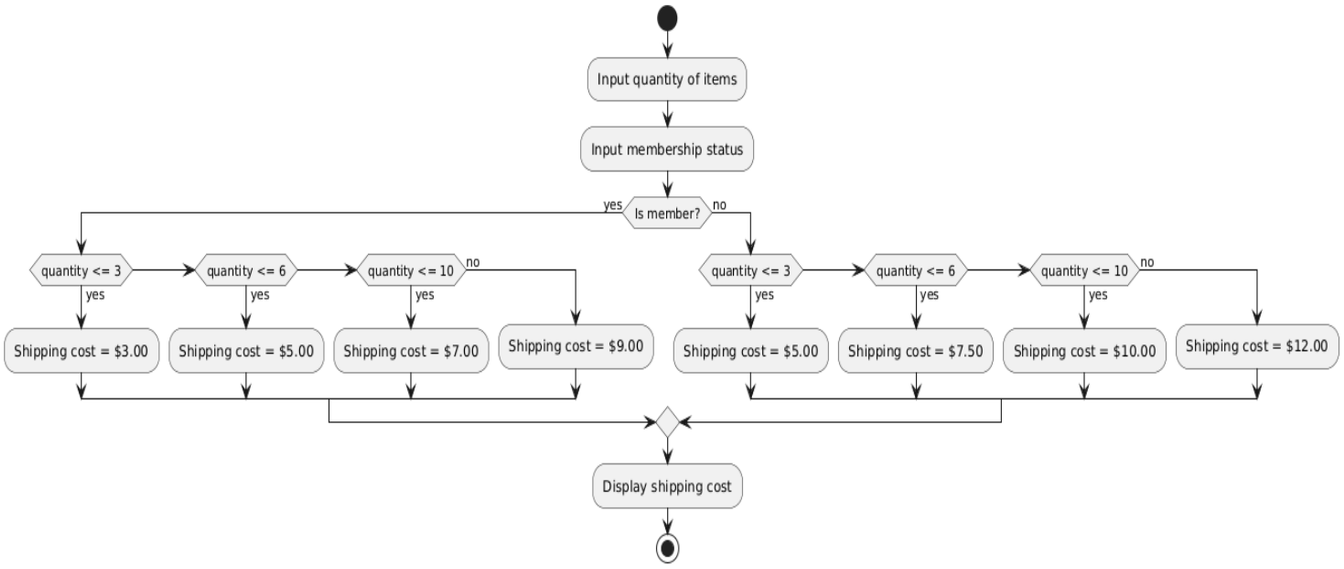
```
DECLARE
    lv_item_price NUMBER(10,2) := 25.00;
    lv_total_available NUMBER(10,2) := 100.00;
    lv_items_purchased NUMBER := 0;
    lv_total_spent NUMBER(10,2) := 0;
BEGIN
    LOOP
        EXIT WHEN lv_total_spent + lv_item_price > lv_total_available;
        lv_items_purchased := lv_items_purchased + 1;
        lv_total_spent := lv_total_spent + lv_item_price;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Items purchased: ' || lv_items_purchased);
    DBMS_OUTPUT.PUT_LINE('Total spent: $' || TO_CHAR(lv_total_spent, '999.99'));
END;
```



**Question 2-7: Creating a Flowchart**

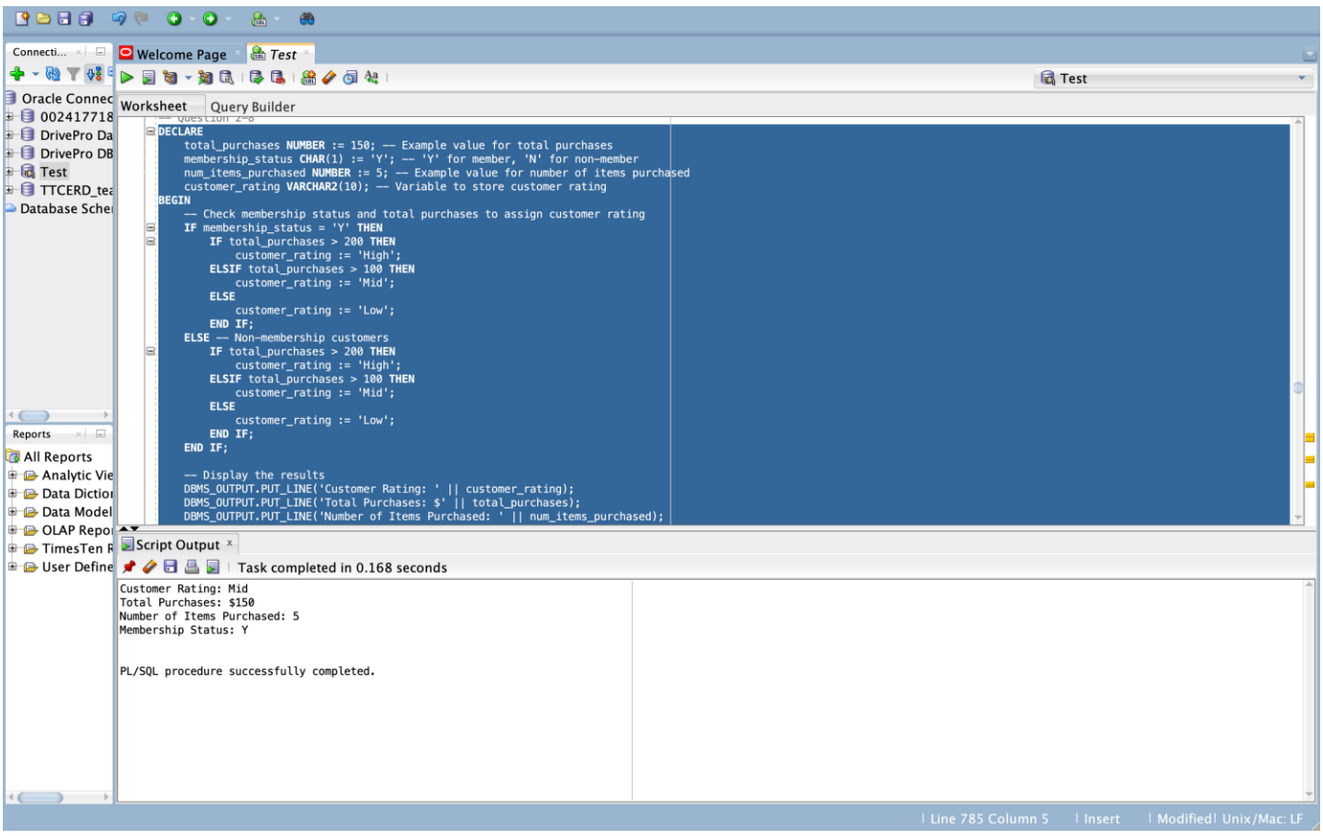
Brewbean's determines shipping costs based on the number of items ordered and club membership status. The applicable rates are shown in the following chart. Develop a flow chart to outline the condition-processing steps needed to handle this calculation.

| Quantity of Items | Nonmember Shipping Cost | Member Shipping Cost |
|---|---|---|
| Up to 3 | $5.00 | $3.00 |
| 4–6 | $7.50 | $5.00 |
| 7–10 | $10.00 | $7.00 |
| More than 10 | $12.00 | $9.00 |



Question 2-8: Using IF Statements
Create a block to accomplish the task outlined in Assignment 2-2. Include a variable containing a Y or N to indicate membership status and a variable to represent the number of items purchased. Test with a variety of values.

```
DECLARE
    total_purchases NUMBER := 150; -- Example value for total purchases
    membership_status CHAR(1) := 'Y'; -- 'Y' for member, 'N' for non-member
    num_items_purchased NUMBER := 5; -- Example value for number of items purchased
    customer_rating VARCHAR2(10); -- Variable to store customer rating
BEGIN
    -- Check membership status and total purchases to assign customer rating
    IF membership_status = 'Y' THEN
        IF total_purchases > 200 THEN
            customer_rating := 'High';
        ELSIF total_purchases > 100 THEN
            customer_rating := 'Mid';
        ELSE
            customer_rating := 'Low';
        END IF;
    ELSE -- Non-membership customers
        IF total_purchases > 200 THEN
            customer_rating := 'High';
        ELSIF total_purchases > 100 THEN
            customer_rating := 'Mid';
        ELSE
            customer_rating := 'Low';
        END IF;
    END IF;

    -- Display the results
    DBMS_OUTPUT.PUT_LINE('Customer Rating: ' || customer_rating);
    DBMS_OUTPUT.PUT_LINE('Total Purchases: $' || total_purchases);
    DBMS_OUTPUT.PUT_LINE('Number of Items Purchased: ' || num_items_purchased);
    DBMS_OUTPUT.PUT_LINE('Membership Status: ' || membership_status);
END;
```

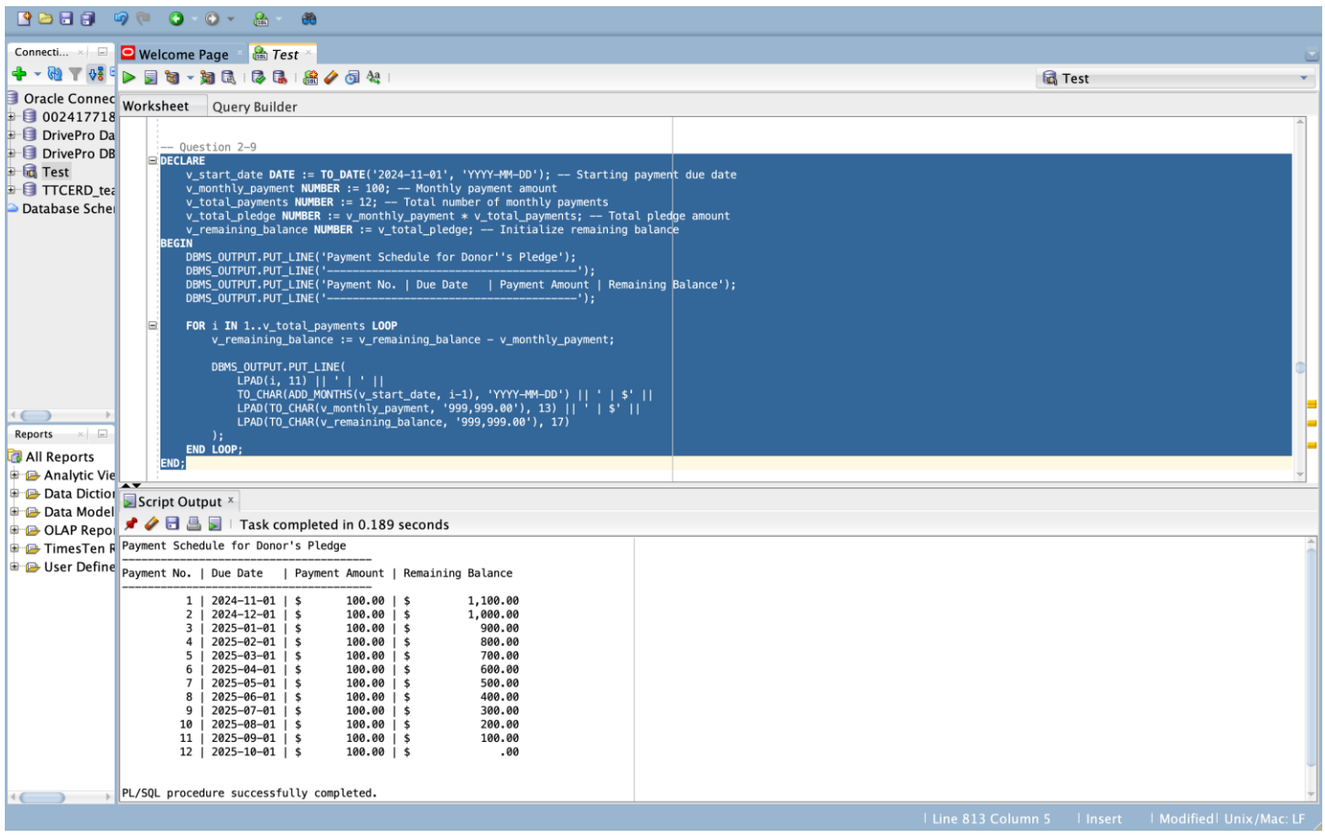



* Change values in ‘DECLARE’ block and the output will vary giving correct answers for each category.

Question 2-9: Using for Loop

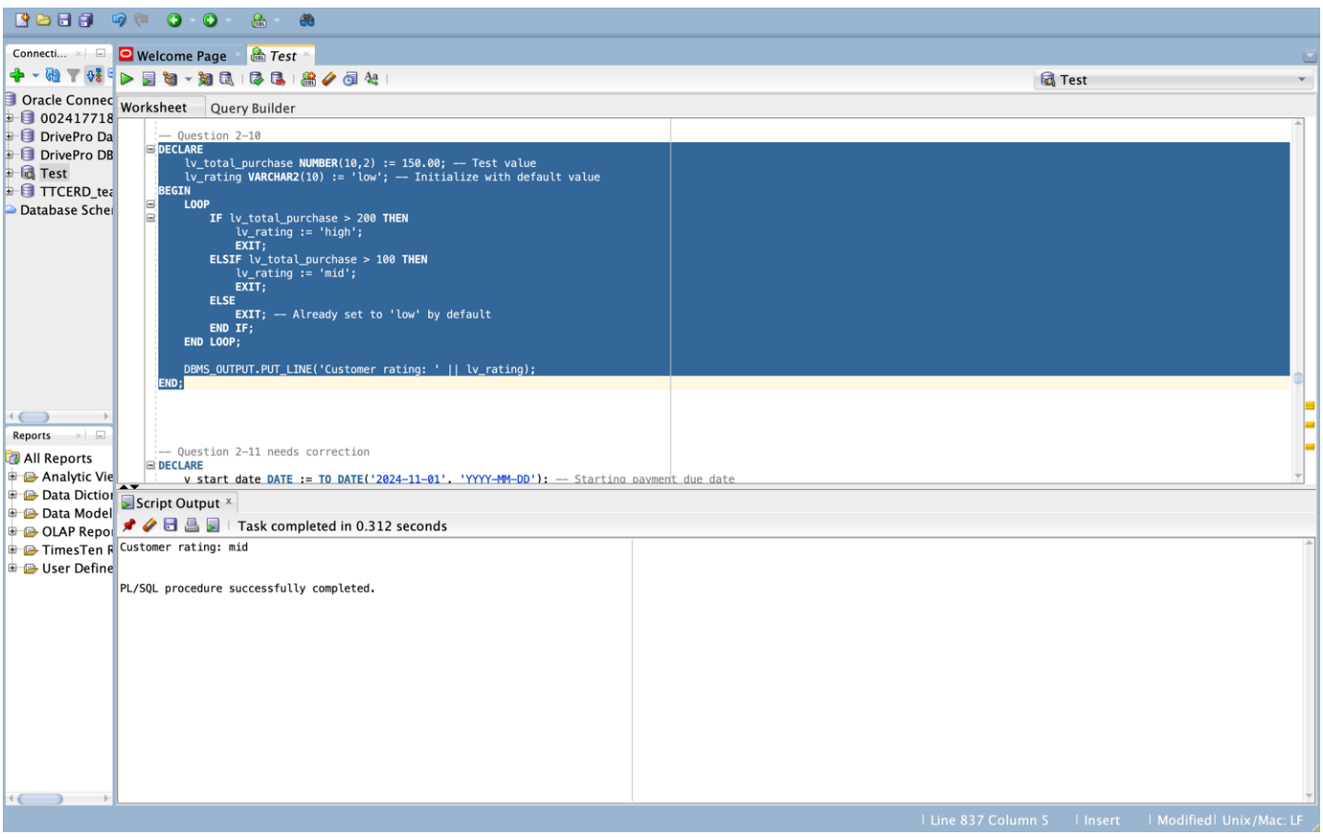
Create a PL/SQL block using FOR loop to generate a payment schedule for a donor’s pledge, which is to be paid monthly in equal increments. Values available for the block are starting payment due date, monthly payment amount, and number of total monthly payments for the pledge. The list that’s generated should display a line for each monthly payment showing payment number, date due, payment amount, and donation balance (remaining amount of pledge owed).

```
DECLARE
    v_start_date DATE := TO_DATE('2024-11-01', 'YYYY-MM-DD'); -- Starting payment due date
    v_monthly_payment NUMBER := 100; -- Monthly payment amount
    v_total_payments NUMBER := 12; -- Total number of monthly payments
    v_total_pledge NUMBER := v_monthly_payment * v_total_payments; -- Total pledge amount
    v_remaining_balance NUMBER := v_total_pledge; -- Initialize remaining balance
BEGIN
    DBMS_OUTPUT.PUT_LINE('Payment Schedule for Donor"s Pledge');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Payment No. | Due Date   | Payment Amount | Remaining Balance');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');

    FOR i IN 1..v_total_payments LOOP
        v_remaining_balance := v_remaining_balance - v_monthly_payment;

        DBMS_OUTPUT.PUT_LINE(
            LPAD(i, 11) || ' | ' ||
            TO_CHAR(ADD_MONTHS(v_start_date, i-1), 'YYYY-MM-DD') || ' | $' ||
            LPAD(TO_CHAR(v_monthly_payment, '999,999.00'), 13) || ' | $' ||
            LPAD(TO_CHAR(v_remaining_balance, '999,999.00'), 17)
        );
    END LOOP;
END;
```


Welcome Page
Test
Worksheet
Query Builder
-- Question 2-9
DECLARE
v_start_date DATE := TO_DATE('2024-11-01', 'YYYY-MM-DD'); -- Starting payment due date
v_monthly_payment NUMBER := 100; -- Monthly payment amount
v_total_payments NUMBER := 12; -- Total number of monthly payments
v_total_pledge NUMBER := v_monthly_payment * v_total_payments; -- Total pledge amount
v_remaining_balance NUMBER := v_total_pledge; -- Initialize remaining balance
BEGIN
DBMS_OUTPUT.PUT_LINE('Payment Schedule for Donor''s Pledge');
DBMS_OUTPUT.PUT_LINE('----------------------------------------');
DBMS_OUTPUT.PUT_LINE('Payment No. | Due Date | Payment Amount | Remaining Balance');
DBMS_OUTPUT.PUT_LINE('----------------------------------------');
FOR i IN 1..v_total_payments LOOP
v_remaining_balance := v_remaining_balance - v_monthly_payment;
DBMS_OUTPUT.PUT_LINE(
LPAD(i, 11) || ' | ' ||
TO_CHAR(ADD_MONTHS(v_start_date, i-1), 'YYYY-MM-DD') || ' | $' ||
LPAD(TO_CHAR(v_monthly_payment, '999,999.00'), 13) || ' | $' ||
LPAD(TO_CHAR(v_remaining_balance, '999,999.00'), 17)
);
END LOOP;
END;
Script Output
Task completed in 0.189 seconds
Payment Schedule for Donor's Pledge
Payment No. | Due Date | Payment Amount | Remaining Balance
1 | 2024-11-01 | $ 100.00 | $ 1,100.00
2 | 2024-12-01 | $ 100.00 | $ 1,000.00
3 | 2025-01-01 | $ 100.00 | $ 900.00
4 | 2025-02-01 | $ 100.00 | $ 800.00
5 | 2025-03-01 | $ 100.00 | $ 700.00
6 | 2025-04-01 | $ 100.00 | $ 600.00
7 | 2025-05-01 | $ 100.00 | $ 500.00
8 | 2025-06-01 | $ 100.00 | $ 400.00
9 | 2025-07-01 | $ 100.00 | $ 300.00
10 | 2025-08-01 | $ 100.00 | $ 200.00
11 | 2025-09-01 | $ 100.00 | $ 100.00
12 | 2025-10-01 | $ 100.00 | $ .00
PL/SQL procedure successfully completed.
Line 813 Column 5
Insert
Modified
Unix/Mac: LF

Question 2-10: Using a BASIC Loop
Accomplish the task in Assignment 2-4 by using a basic loop structure.

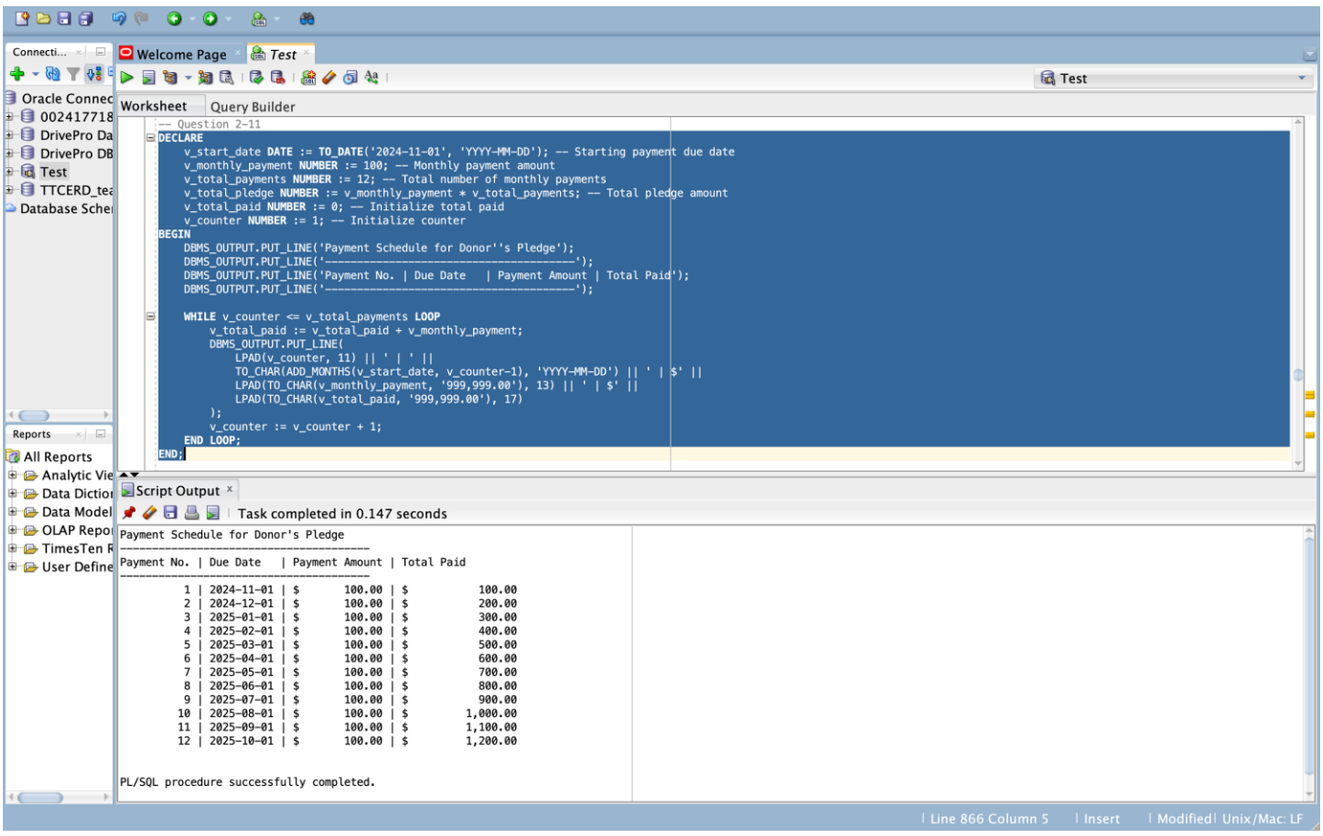
```
DECLARE
    lv_total_purchase NUMBER(10,2) := 150.00; -- Test value
    lv_rating VARCHAR2(10) := 'low'; -- Initialize with default value
BEGIN
    LOOP
        IF lv_total_purchase > 200 THEN
            lv_rating := 'high';
            EXIT;
        ELSIF lv_total_purchase > 100 THEN
            lv_rating := 'mid';
            EXIT;
        ELSE
            EXIT; -- Already set to 'low' by default
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Customer rating: ' || lv_rating);
END;
```




* Change the test value to get rating of each category correctly.

Question 2-11: Using a WHILE Loop
Accomplish the task in Assignment 2-4 by using a While loop structure. Instead of displaying the donation balance (remaining amount of pledge owed) on each line of output, display the total paid to date.

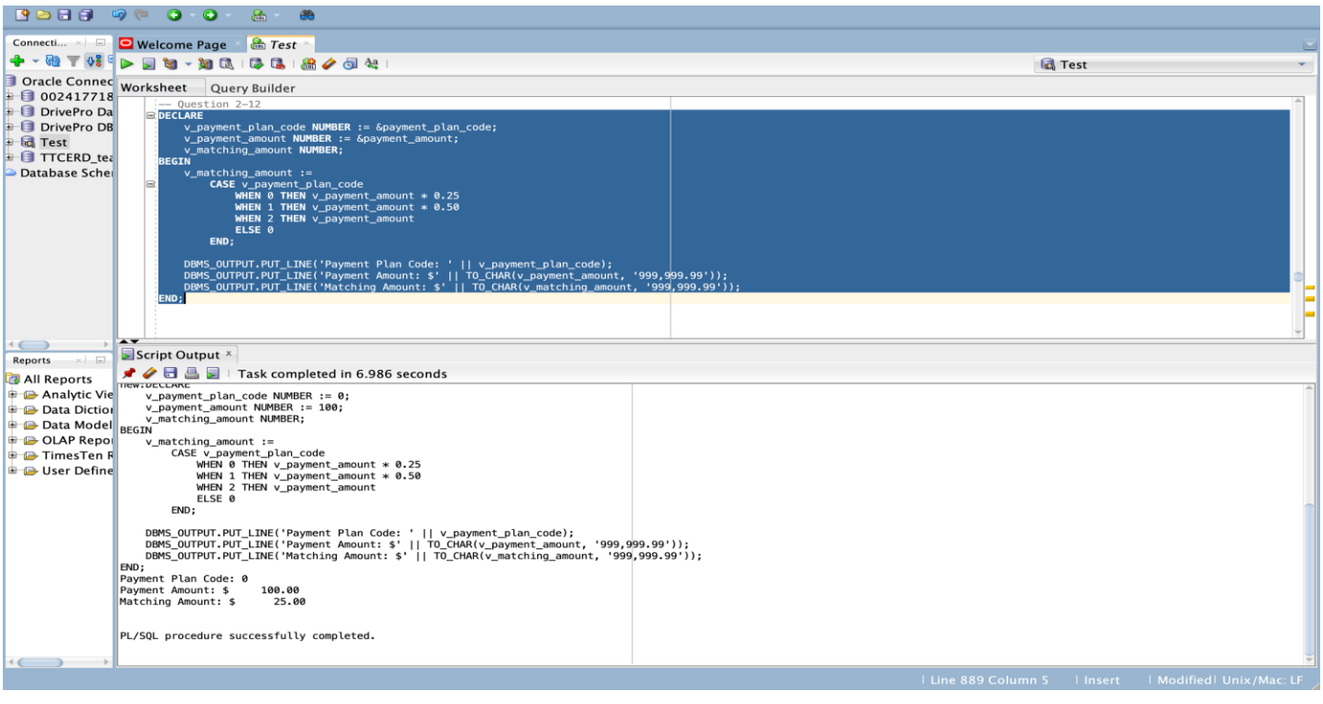
```
DECLARE
    v_start_date DATE := TO_DATE('2024-11-01', 'YYYY-MM-DD'); -- Starting payment due date
    v_monthly_payment NUMBER := 100; -- Monthly payment amount
    v_total_payments NUMBER := 12; -- Total number of monthly payments
    v_total_pledge NUMBER := v_monthly_payment * v_total_payments; -- Total pledge amount
    v_total_paid NUMBER := 0; -- Initialize total paid
    v_counter NUMBER := 1; -- Initialize counter
BEGIN
    DBMS_OUTPUT.PUT_LINE('Payment Schedule for Donor"s Pledge');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Payment No. | Due Date   | Payment Amount | Total Paid');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');

    WHILE v_counter <= v_total_payments LOOP
        v_total_paid := v_total_paid + v_monthly_payment;
        DBMS_OUTPUT.PUT_LINE(
            LPAD(v_counter, 11) || ' | ' ||
            TO_CHAR(ADD_MONTHS(v_start_date, v_counter-1), 'YYYY-MM-DD') || ' | $' ||
            LPAD(TO_CHAR(v_monthly_payment, '999,999.00'), 13) || ' | $' ||
            LPAD(TO_CHAR(v_total_paid, '999,999.00'), 17)
        );
        v_counter := v_counter + 1;
    END LOOP;
END;
```


Welcome Page
Test
Worksheet
Query Builder
-- Question 2-11
DECLARE
v_start_date DATE := TO_DATE('2024-11-01', 'YYYY-MM-DD'); -- Starting payment due date
v_monthly_payment NUMBER := 100; -- Monthly payment amount
v_total_payments NUMBER := 12; -- Total number of monthly payments
v_total_pledge NUMBER := v_monthly_payment * v_total_payments; -- Total pledge amount
v_total_paid NUMBER := 0; -- Initialize total paid
v_counter NUMBER := 1; -- Initialize counter
BEGIN
DBMS_OUTPUT.PUT_LINE('Payment Schedule for Donor''s Pledge');
DBMS_OUTPUT.PUT_LINE('----------------------------------------');
DBMS_OUTPUT.PUT_LINE('Payment No. | Due Date | Payment Amount | Total Paid');
DBMS_OUTPUT.PUT_LINE('----------------------------------------');
WHILE v_counter <= v_total_payments LOOP
v_total_paid := v_total_paid + v_monthly_payment;
DBMS_OUTPUT.PUT_LINE(
LPAD(v_counter, 11) || ' | ' ||
TO_CHAR(ADD_MONTHS(v_start_date, v_counter-1), 'YYYY-MM-DD') || ' | $' ||
LPAD(TO_CHAR(v_monthly_payment, '999,999.00'), 13) || ' | $' ||
LPAD(TO_CHAR(v_total_paid, '999,999.00'), 17)
);
v_counter := v_counter + 1;
END LOOP;
END;
Script Output
Task completed in 0.147 seconds
Payment Schedule for Donor's Pledge
Payment No. | Due Date | Payment Amount | Total Paid
1 | 2024-11-01 | $ 100.00 | $ 100.00
2 | 2024-12-01 | $ 100.00 | $ 200.00
3 | 2025-01-01 | $ 100.00 | $ 300.00
4 | 2025-02-01 | $ 100.00 | $ 400.00
5 | 2025-03-01 | $ 100.00 | $ 500.00
6 | 2025-04-01 | $ 100.00 | $ 600.00
7 | 2025-05-01 | $ 100.00 | $ 700.00
8 | 2025-06-01 | $ 100.00 | $ 800.00
9 | 2025-07-01 | $ 100.00 | $ 900.00
10 | 2025-08-01 | $ 100.00 | $ 1,000.00
11 | 2025-09-01 | $ 100.00 | $ 1,100.00
12 | 2025-10-01 | $ 100.00 | $ 1,200.00
PL/SQL procedure successfully completed.
Line 866 Column 5
Insert
Modified
Unix/Mac: LF

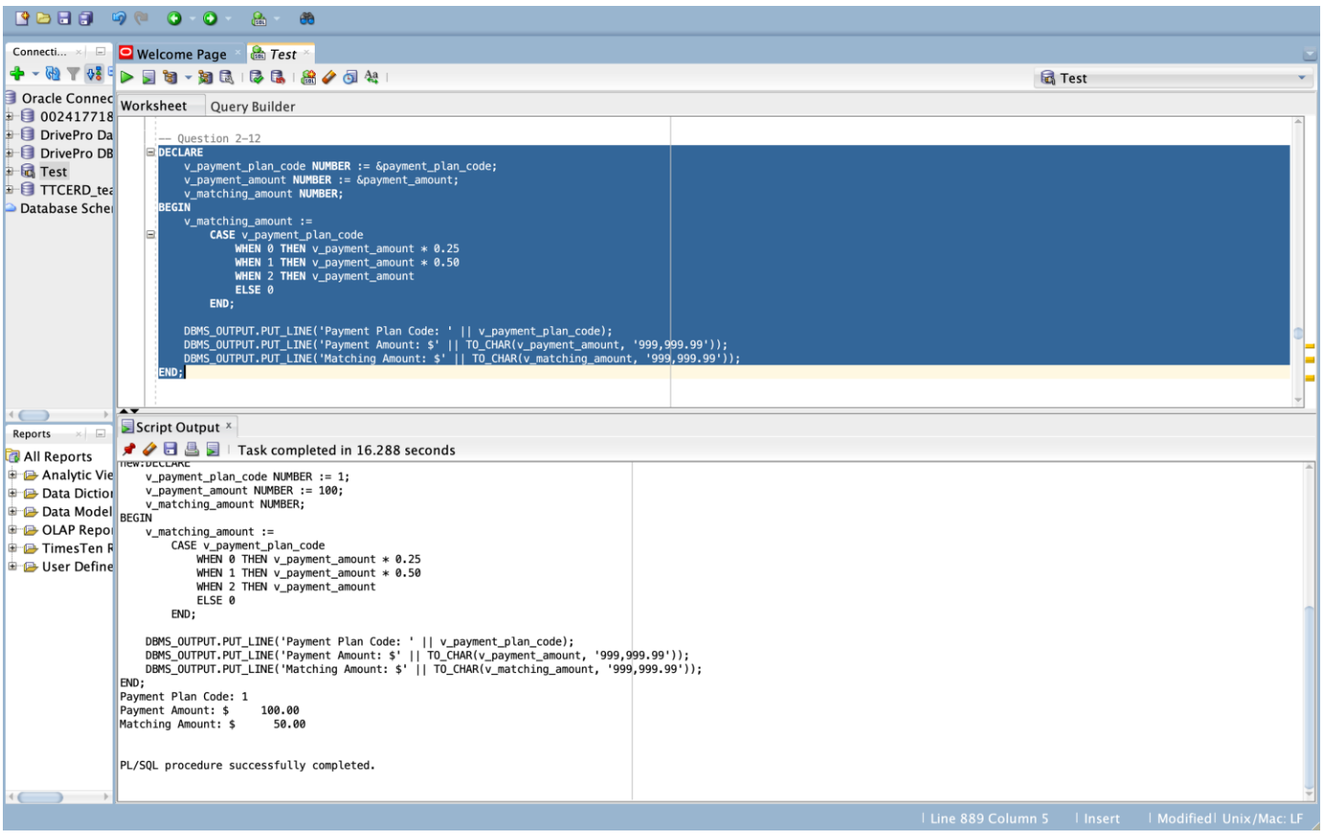
Question 2-12: Using a CASE Expression
Donors can select one of three payment plans for a pledge indicated by the following codes:0 = one-time (lump sum) payment, 1 = monthly payments over one year, and 2 = monthly payments over two years. A local business has agreed to pay matching amounts on pledge payments during the current month. A PL/SQL block is needed to identify the matching amount for a pledge payment. Create a block using input values of a payment plan code and a payment amount. Use a CASE expression to calculate the matching amount, based on the payment plan codes 0 = 25%, 1= 50%, 2 = 100%, and other = 0. Display the calculated amount.

```
DECLARE
   v_payment_plan_code NUMBER := &payment_plan_code;
   v_payment_amount NUMBER := &payment_amount;
   v_matching_amount NUMBER;
BEGIN
   v_matching_amount :=
      CASE v_payment_plan_code
         WHEN 0 THEN v_payment_amount * 0.25
         WHEN 1 THEN v_payment_amount * 0.50
         WHEN 2 THEN v_payment_amount
         ELSE 0
      END;

   DBMS_OUTPUT.PUT_LINE('Payment Plan Code: ' || v_payment_plan_code);
   DBMS_OUTPUT.PUT_LINE('Payment Amount: $' || TO_CHAR(v_payment_amount, '999,999.99'));
   DBMS_OUTPUT.PUT_LINE('Matching Amount: $' || TO_CHAR(v_matching_amount, '999,999.99'));
END;
```

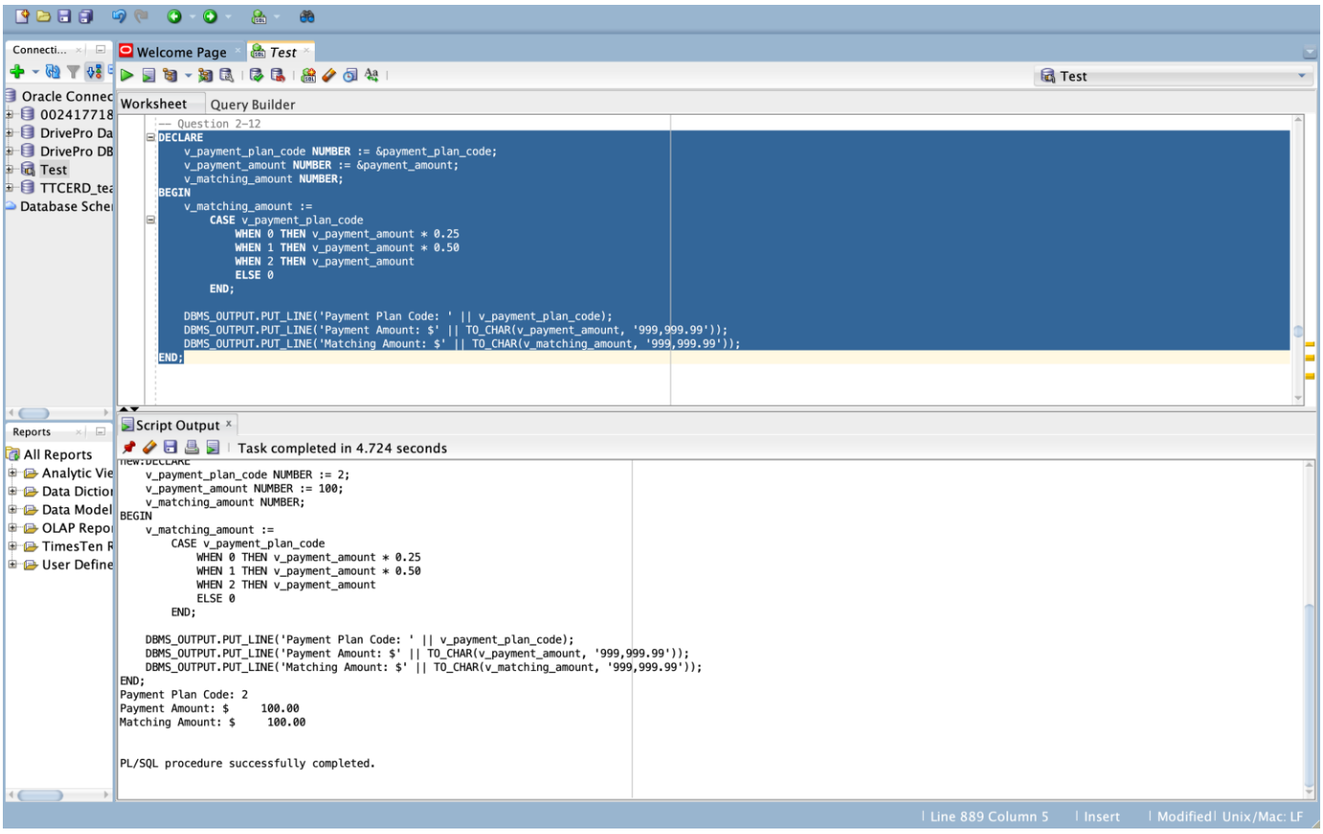
Payment plan : 0



Payment plan : 1




Payment plan : 2



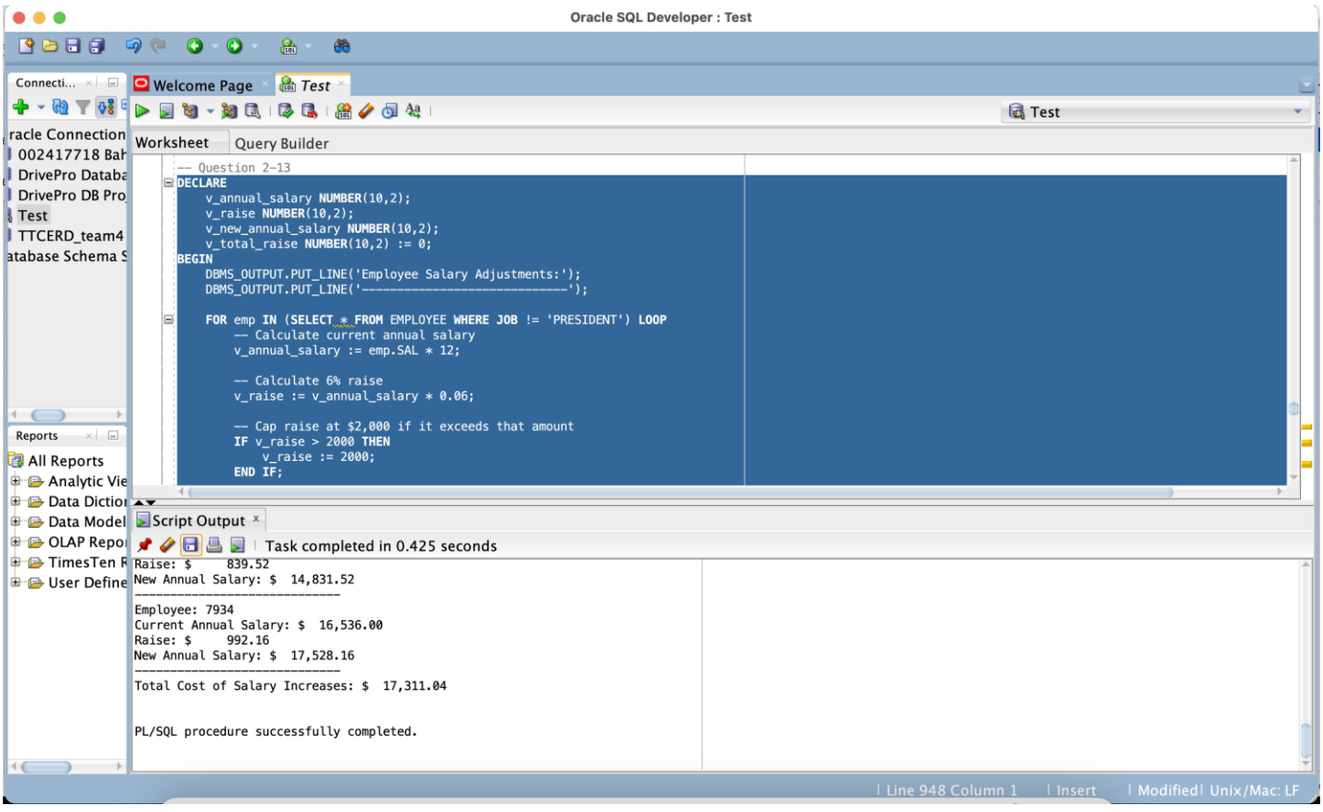
**Question 2-13: Processing and Updating a Group of Rows**

To help track employee information, a new EMPLOYEE table was added to the Brewbean's database. Review the data in this table. A PL/SQL block is needed to calculate annual raises and update employee salary amounts in the table. Create a block that addresses all the requirements in the following list. All salaries in the EMPLOYEE table are recorded as monthly amounts. *Tip*: Display the calculated salaries for verification before including the update action.

- Calculate 6% annual raises for all employees except the president.
- If a 6% raise totals more than $2,000, cap the raise at $2,000.
- Update the salary for each employee in the table.
- For each employee number, display the current annual salary, raise, and proposed new annual salary.
- Finally, following the details for each employee, show the total cost of all employees' salary increases for Brewbean's.

```
DECLARE
   v_annual_salary NUMBER(10,2);
   v_raise NUMBER(10,2);
   v_new_annual_salary NUMBER(10,2);
   v_total_raise NUMBER(10,2) := 0;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Employee Salary Adjustments:');
   DBMS_OUTPUT.PUT_LINE('-----------------------------');

   FOR emp IN (SELECT * FROM EMPLOYEE WHERE JOB != 'PRESIDENT') LOOP
      -- Calculate current annual salary
      v_annual_salary := emp.SAL * 12;

      -- Calculate 6% raise
      v_raise := v_annual_salary * 0.06;

      -- Cap raise at $2,000 if it exceeds that amount
      IF v_raise > 2000 THEN
         v_raise := 2000;
      END IF;

      -- Calculate new annual salary
      v_new_annual_salary := v_annual_salary + v_raise;

      -- Display employee information
      DBMS_OUTPUT.PUT_LINE('Employee: ' || emp.EMPNO);
```

```
        DBMS_OUTPUT.PUT_LINE('Current Annual Salary: $' || TO_CHAR(v_annual_salary, '999,999.99'));
        DBMS_OUTPUT.PUT_LINE('Raise: $' || TO_CHAR(v_raise, '999,999.99'));
        DBMS_OUTPUT.PUT_LINE('New Annual Salary: $' || TO_CHAR(v_new_annual_salary, '999,999.99'));
        DBMS_OUTPUT.PUT_LINE('----------------------------');

        -- Update employee salary in the table
        UPDATE EMPLOYEE
        SET SAL = v_new_annual_salary / 12
        WHERE EMPNO = emp.EMPNO;

        -- Add to total raise amount
        v_total_raise := v_total_raise + v_raise;
    END LOOP;

    -- Display total cost of salary increases
    DBMS_OUTPUT.PUT_LINE('Total Cost of Salary Increases: $' || TO_CHAR(v_total_raise, '999,999.99'));

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK;
END;
```

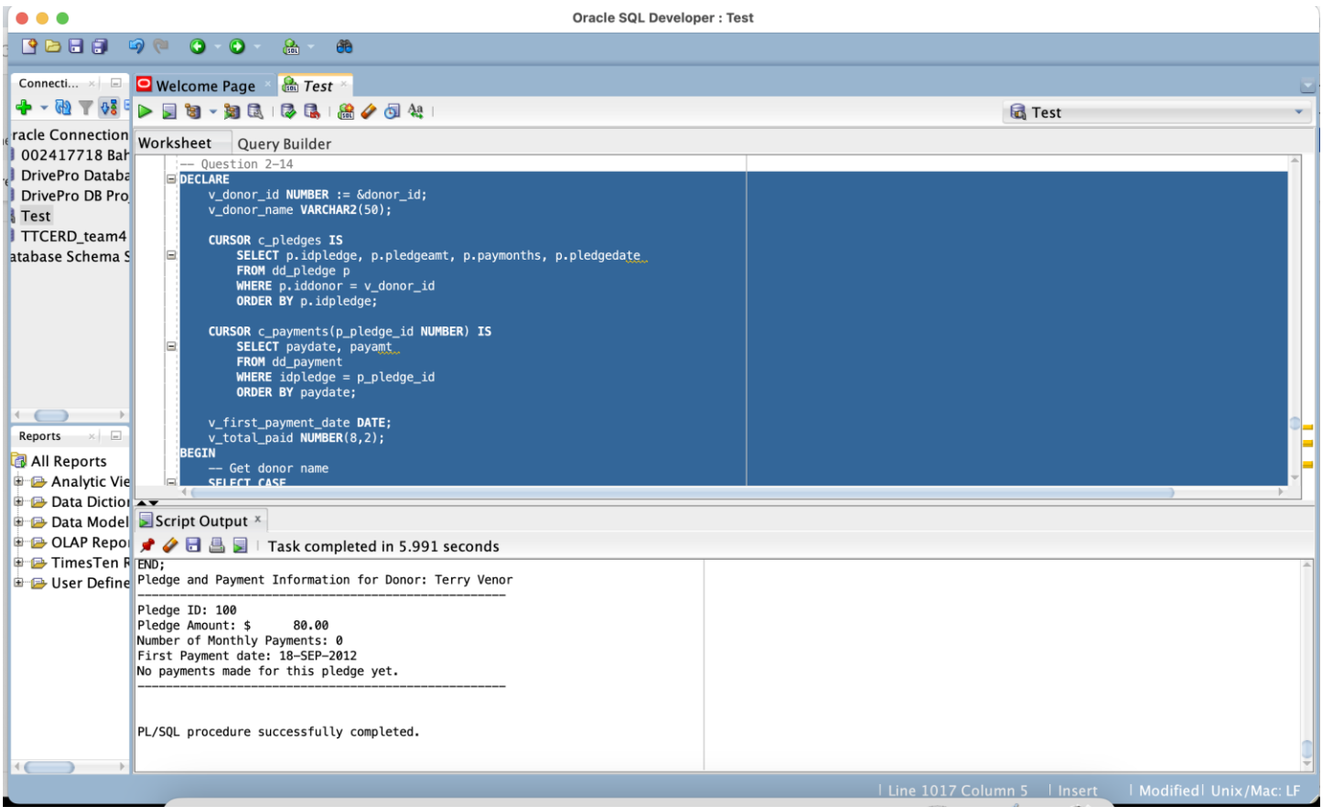


```
Employee Salary Adjustments:
--------------------------------
Employee: 7698
Current Annual Salary: $  36,200.04
Raise: $   2,000.00
New Annual Salary: $  38,200.04
--------------------------------
Employee: 7782
Current Annual Salary: $  31,164.00
Raise: $   1,869.84
New Annual Salary: $  33,033.84
--------------------------------
Employee: 7566
Current Annual Salary: $  37,700.04
Raise: $   2,000.00
New Annual Salary: $  39,700.04
--------------------------------
Employee: 7654
Current Annual Salary: $  15,900.00
Raise: $     954.00
New Annual Salary: $  16,854.00
--------------------------------
Employee: 7499
Current Annual Salary: $  20,352.00
Raise: $   1,221.12
New Annual Salary: $  21,573.12
--------------------------------
Employee: 7844
Current Annual Salary: $  19,080.00
Raise: $   1,144.80
New Annual Salary: $  20,224.80
--------------------------------
Employee: 7900
Current Annual Salary: $  12,084.00
Raise: $     725.04
New Annual Salary: $  12,809.04
--------------------------------
Employee: 7521
Current Annual Salary: $  15,900.00
Raise: $     954.00
New Annual Salary: $  16,854.00
--------------------------------
Employee: 7902
Current Annual Salary: $  38,000.04
Raise: $   2,000.00
New Annual Salary: $  40,000.04
--------------------------------
Employee: 7369
Current Annual Salary: $  10,176.00
Raise: $     610.56
New Annual Salary: $  10,786.56
--------------------------------
Employee: 7788
Current Annual Salary: $  38,000.04
Raise: $   2,000.00
New Annual Salary: $  40,000.04
--------------------------------
Employee: 7876
Current Annual Salary: $  13,992.00
Raise: $     839.52
New Annual Salary: $  14,831.52
--------------------------------
Employee: 7934
Current Annual Salary: $  16,536.00
Raise: $     992.16
New Annual Salary: $  17,528.16
--------------------------------
Total Cost of Salary Increases: $  17,311.04
```

Question 2-14: Using an Explicit Cursor:
Create a block to retrieve and display pledge and payment information for a specific donor. Foreach pledge payment from the donor, display the pledge ID, pledge amount, number of monthly payments, payment date, and payment amount. The list should be sorted by pledge ID and then by payment date. For the first payment made for each pledge, display “first payment” on that output row.

```
DECLARE
   v_donor_id NUMBER := &donor_id;
   v_donor_name VARCHAR2(50);

   CURSOR c_pledges IS
      SELECT p.idpledge, p.pledgeamt, p.paymonths, p.pledgedate
      FROM dd_pledge p
      WHERE p.iddonor = v_donor_id
      ORDER BY p.idpledge;

   CURSOR c_payments(p_pledge_id NUMBER) IS
      SELECT paydate, payamt
      FROM dd_payment
      WHERE idpledge = p_pledge_id
      ORDER BY paydate;

   v_first_payment_date DATE;
   v_total_paid NUMBER(8,2);
BEGIN
   -- Get donor name
   SELECT CASE
         WHEN firstname IS NOT NULL THEN firstname || ' ' || lastname
         ELSE lastname
         END INTO v_donor_name
   FROM dd_donor
   WHERE iddonor = v_donor_id;

   DBMS_OUTPUT.PUT_LINE('Pledge and Payment Information for Donor: ' || v_donor_name);
   DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');

   FOR pledge_rec IN c_pledges LOOP
      DBMS_OUTPUT.PUT_LINE('Pledge ID: ' || pledge_rec.idpledge);
      DBMS_OUTPUT.PUT_LINE('Pledge Amount: $' || TO_CHAR(pledge_rec.pledgeamt, '999,999.99'));
```

```
        DBMS_OUTPUT.PUT_LINE('Number of Monthly Payments: ' || NVL(pledge_rec.paymonths, 0));
        DBMS_OUTPUT.PUT_LINE('First Payment date: ' || TO_CHAR(pledge_rec.pledgedate, 'DD-MON-YYYY'));

        -- Get the first payment date and total paid amount
        SELECT MIN(paydate), NVL(SUM(payamt), 0)
        INTO v_first_payment_date, v_total_paid
        FROM dd_payment
        WHERE idpledge = pledge_rec.idpledge;

        IF v_first_payment_date IS NOT NULL THEN
            DBMS_OUTPUT.PUT_LINE('First Payment Date: ' || TO_CHAR(v_first_payment_date, 'DD-MON-YYYY'));
            DBMS_OUTPUT.PUT_LINE('Total Amount Paid: $' || TO_CHAR(v_total_paid, '999,999.99'));
            DBMS_OUTPUT.PUT_LINE('Payments:');

            FOR payment_rec IN c_payments(pledge_rec.idpledge) LOOP
                DBMS_OUTPUT.PUT_LINE('  Date: ' || TO_CHAR(payment_rec.paydate, 'DD-MON-YYYY') ||
                                     ', Amount: $' || TO_CHAR(payment_rec.payamt, '999,999.99') ||
                                     CASE WHEN payment_rec.paydate = v_first_payment_date THEN ' (First Payment)' ELSE '' END);
            END LOOP;
        ELSE
            DBMS_OUTPUT.PUT_LINE('No payments made for this pledge yet.');
        END IF;

        DBMS_OUTPUT.PUT_LINE('--------------------------------------------------');
    END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No donor found with ID ' || v_donor_id);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

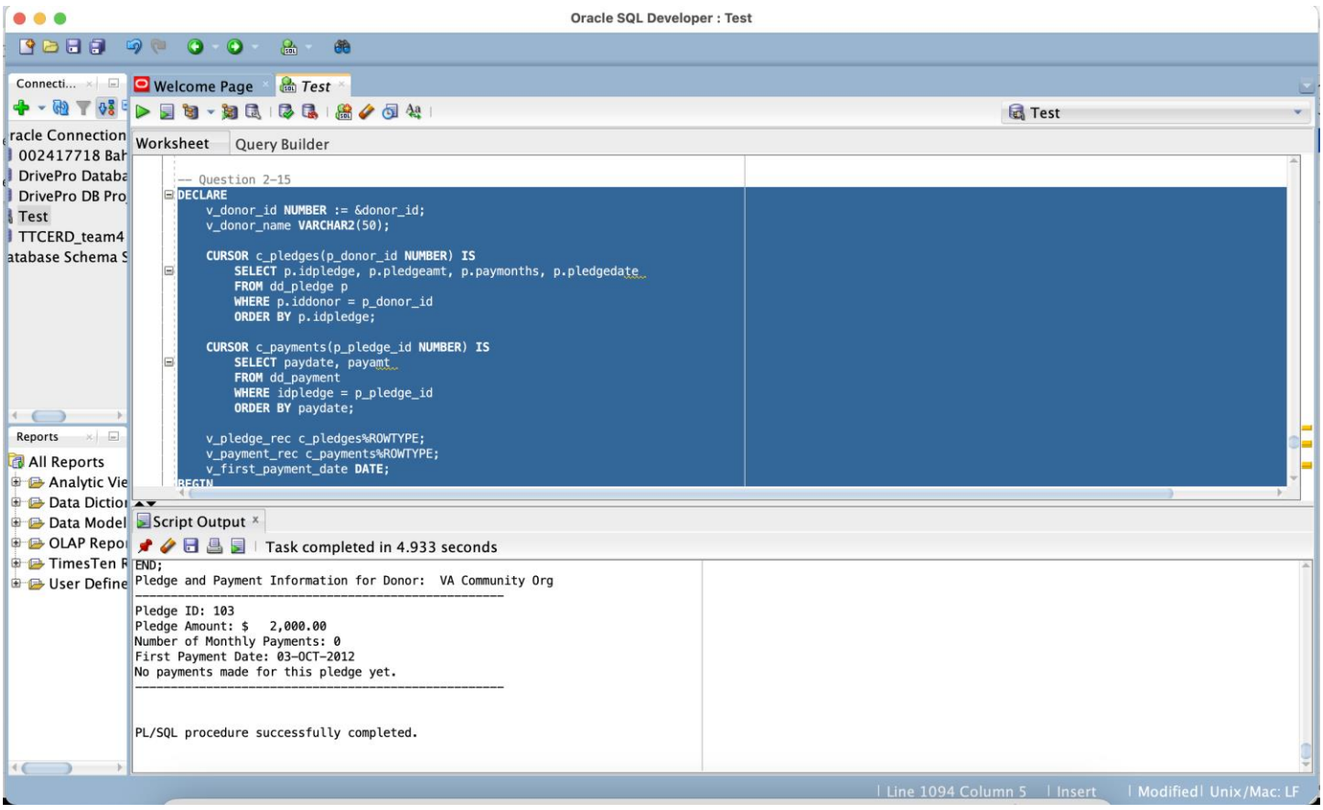



* After running the script for 2-14, it will ask for donor ID, here for demonstration ID303 is used. You can enter any ID and check the output accordingly.

Question 2-15: Using a Different Form of Explicit Cursors: Redo Above question but use a different cursor form to perform the same task

```
DECLARE
    v_donor_id NUMBER := &donor_id;
    v_donor_name VARCHAR2(50);

    CURSOR c_pledges(p_donor_id NUMBER) IS
        SELECT p.idpledge, p.pledgeamt, p.paymonths, p.pledgedate
        FROM dd_pledge p
        WHERE p.iddonor = p_donor_id
        ORDER BY p.idpledge;

    CURSOR c_payments(p_pledge_id NUMBER) IS
        SELECT paydate, payamt
        FROM dd_payment
        WHERE idpledge = p_pledge_id
        ORDER BY paydate;

    v_pledge_rec c_pledges%ROWTYPE;
    v_payment_rec c_payments%ROWTYPE;
    v_first_payment_date DATE;
BEGIN
    -- Get donor name
    SELECT firstname || ' ' || lastname INTO v_donor_name
    FROM dd_donor
    WHERE iddonor = v_donor_id;

    DBMS_OUTPUT.PUT_LINE('Pledge and Payment Information for Donor: ' || v_donor_name);
    DBMS_OUTPUT.PUT_LINE('--------------------------------------------------');

    OPEN c_pledges(v_donor_id);
    LOOP
        FETCH c_pledges INTO v_pledge_rec;
        EXIT WHEN c_pledges%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Pledge ID: ' || v_pledge_rec.idpledge);
        DBMS_OUTPUT.PUT_LINE('Pledge Amount: $' || TO_CHAR(v_pledge_rec.pledgeamt, '999,999.99'));
```

```
        DBMS_OUTPUT.PUT_LINE('Number of Monthly Payments: ' || NVL(v_pledge_rec.paymonths, 0));
        DBMS_OUTPUT.PUT_LINE('First Payment Date: ' || TO_CHAR(v_pledge_rec.pledgedate, 'DD-MON-YYYY'));

        -- Get the first payment date
        SELECT MIN(paydate) INTO v_first_payment_date
        FROM dd_payment
        WHERE idpledge = v_pledge_rec.idpledge;

        IF v_first_payment_date IS NOT NULL THEN
            DBMS_OUTPUT.PUT_LINE('First Payment Date: ' || TO_CHAR(v_first_payment_date, 'DD-MON-YYYY'));
            DBMS_OUTPUT.PUT_LINE('Payments:');

            OPEN c_payments(v_pledge_rec.idpledge);
            LOOP
                FETCH c_payments INTO v_payment_rec;
                EXIT WHEN c_payments%NOTFOUND;

                DBMS_OUTPUT.PUT_LINE('  Date: ' || TO_CHAR(v_payment_rec.paydate, 'DD-MON-YYYY') ||
                                     ', Amount: $' || TO_CHAR(v_payment_rec.payamt, '999,999.99') ||
                                     CASE WHEN v_payment_rec.paydate = v_first_payment_date THEN ' (First Payment)' ELSE '' END);
            END LOOP;
            CLOSE c_payments;
        ELSE
            DBMS_OUTPUT.PUT_LINE('No payments made for this pledge yet.');
        END IF;

        DBMS_OUTPUT.PUT_LINE('--------------------------------------------------');
    END LOOP;
    CLOSE c_pledges;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No donor found with ID ' || v_donor_id);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

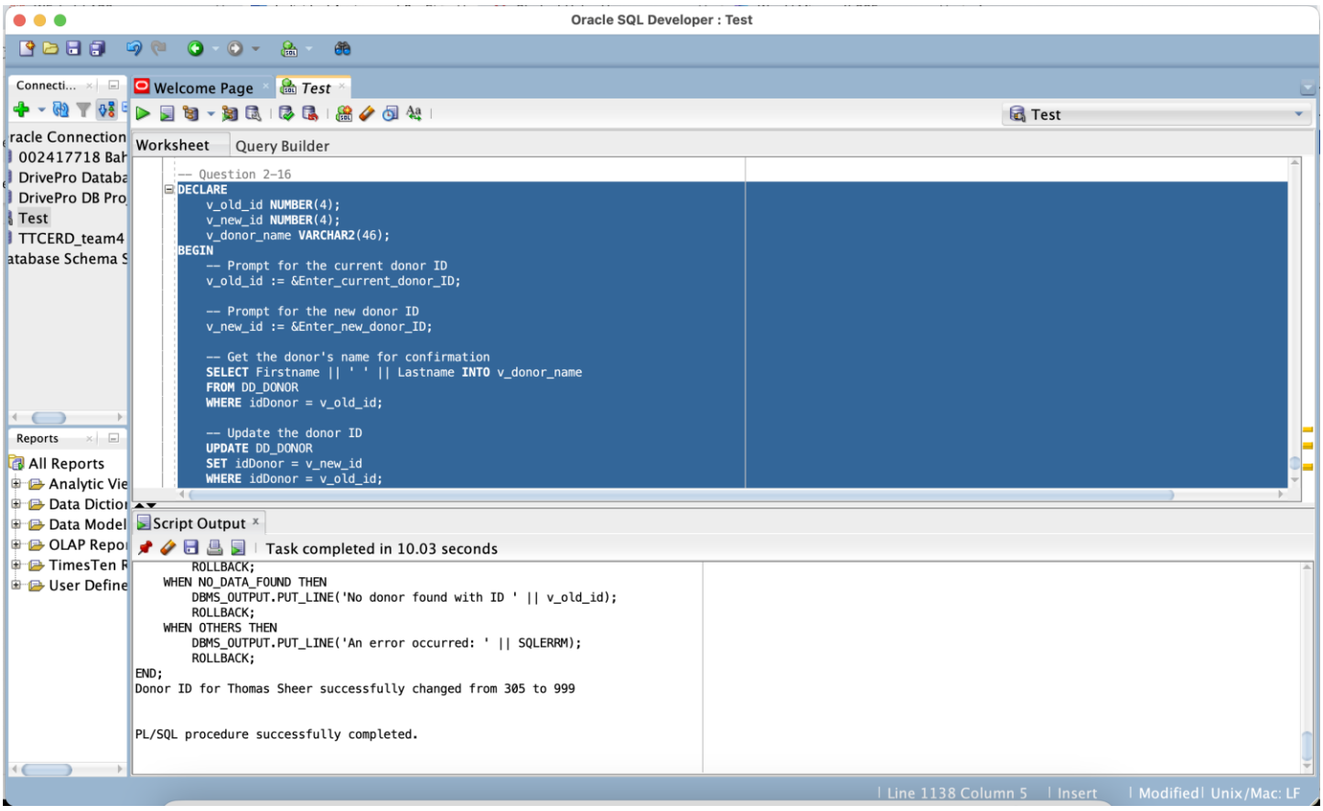



** After running the script for 2-15, it will ask for donor ID, here for demonstration ID307 is used. You can enter any ID and check the output accordingly.

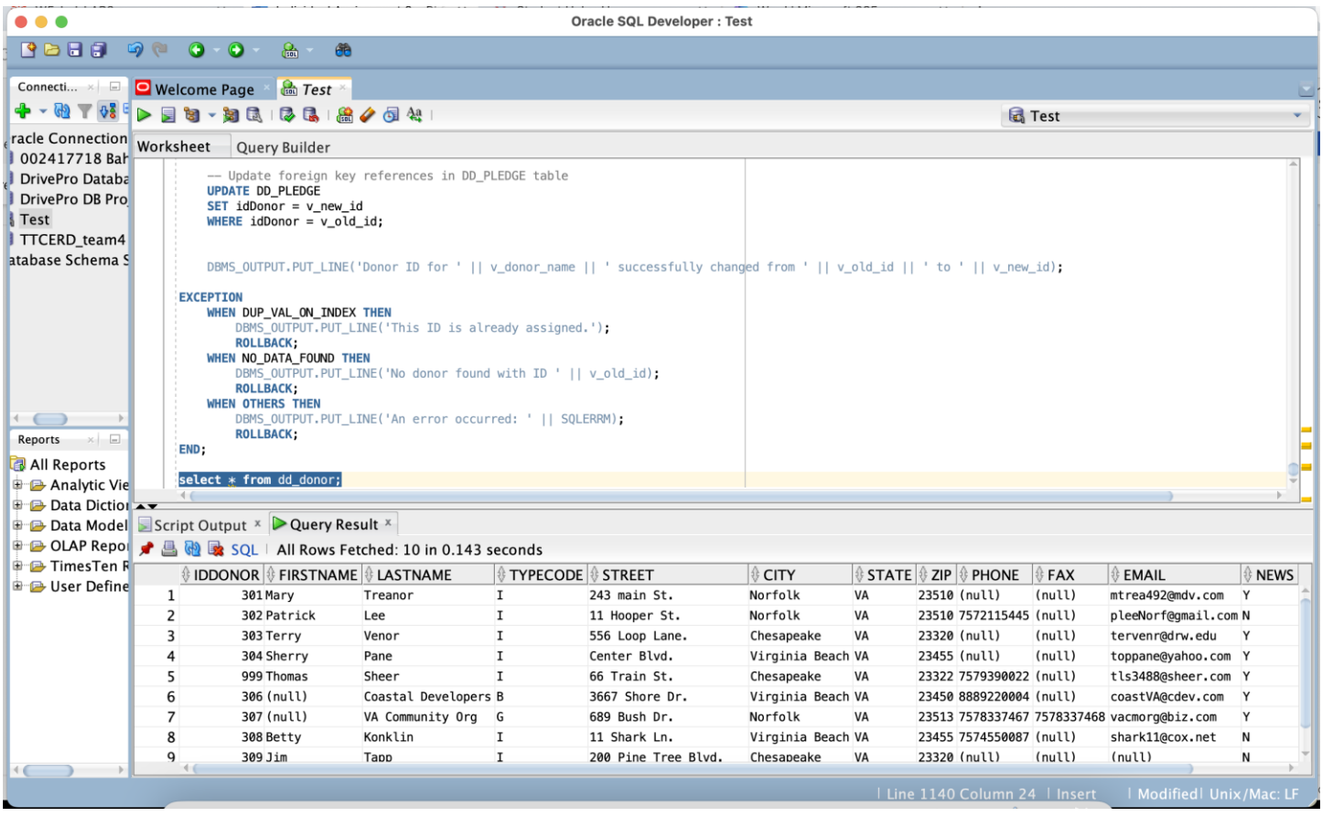
Question 2-16: Exception Handling:
The DoGood Donor application contains a page that allows administrators to change the ID assigned to a donor in the DD_DONOR table. Create a PL/SQL block to handle this task. Include exception-handling code to address an error raised by attempting to enter a duplicate donor ID. If this error occurs, display the message “This ID is already assigned.” Test the code by changing donor ID305. (Don’t include a COMMIT statement; roll back any DML actions use

```
DECLARE
    v_old_id NUMBER(4);
    v_new_id NUMBER(4);
    v_donor_name VARCHAR2(46);
BEGIN
    -- Prompt for the current donor ID
    v_old_id := &Enter_current_donor_ID;

    -- Prompt for the new donor ID
    v_new_id := &Enter_new_donor_ID;

    -- Get the donor's name for confirmation
    SELECT Firstname || ' ' || Lastname INTO v_donor_name
    FROM DD_DONOR
    WHERE idDonor = v_old_id;

    -- Update the donor ID
    UPDATE DD_DONOR
    SET idDonor = v_new_id
    WHERE idDonor = v_old_id;

    -- Update foreign key references in DD_PLEDGE table
    UPDATE DD_PLEDGE
    SET idDonor = v_new_id
    WHERE idDonor = v_old_id;

    DBMS_OUTPUT.PUT_LINE('Donor ID for ' || v_donor_name || ' successfully changed from ' || v_old_id || ' to ' || v_new_id);

EXCEPTION
```

```
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('This ID is already assigned.');
        ROLLBACK;
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No donor found with ID ' || v_old_id);
        ROLLBACK;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK;
END;
```




** ID305 successfully changed to 999, for demonstration purposes.




** ID 305 changed to 999 is reflected in the donor table.