## HANDS ON ASSIGNMENTS PART I

ASSIGNMENT 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 | VAR CHAR2(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 := '10-DECEMBER-12';

lv\_test\_num CONSTANT NUMBER(3) := 10;

lv\_test\_txt VARCHAR2(10) := 'Zuniga';

BEGIN

DBMS\_OUTPUT.PUT\_LINE(lv\_test\_date);

DBMS\_OUTPUT.PUT\_LINE(lv\_test\_num);

DBMS\_OUTPUT.PUT\_LINE(lv\_test\_txt);

END;

ASSIGNMENT 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 conditionals processing steps needed for this block.



ASSIGNMENT 2-3 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(4) := 200;

lv\_cust\_rating VARCHAR2(5);

BEGIN

IF lv\_total\_purchase > 200 THEN

lv\_cust\_rating := 'high';

ELSIF lv\_total\_purchase > 100 THEN

lv\_cust\_rating := 'mid';

ELSE

lv\_cust\_rating := 'low';

END IF;

DBMS\_OUTPUT.PUT\_LINE(lv\_cust\_rating);

END;

ASSIGNMENT 2-4 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 to test your block.

DECLARE

lv\_total\_purchase NUMBER(4) := 300;

lv\_cust\_rating VARCHAR2(5);

BEGIN

CASE

WHEN lv\_total\_purchase > 200 THEN

lv\_cust\_rating := 'high';

WHEN lv\_total\_purchase > 100 THEN

lv\_cust\_rating := 'mid';

ELSE

lv\_cust\_rating := 'low';

END CASE;

DBMS\_OUTPUT.PUT\_LINE(lv\_cust\_rating);

END;

ASSIGNMENT 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(8,2) := 300.50;

lv\_payment\_amount NUMBER(8,2) := 300.50;

lv\_amount\_due BOOLEAN := FALSE;

BEGIN

lv\_account\_balance := lv\_account\_balance - lv\_payment\_amount;

IF lv\_account\_balance > 0 THEN lv\_amount\_due := TRUE;

ELSE lv\_amount\_due := FALSE;

END IF;

IF lv\_amount\_due THEN DBMS\_OUTPUT.PUT\_LINE( lv\_account\_balance );

ELSE DBMS\_OUTPUT.PUT\_LINE('Account is paid in full');

END IF;

END;

ASSIGNMENT 2-6: Using Loop 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 on 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\_price NUMBER(8,2) := 20.00;

lv\_total\_amount NUMBER(8,2) := 200.00;

lv\_total\_spent NUMBER(8,2) := 0;

lv\_number\_items NUMBER(3) := 0;

BEGIN

LOOP

lv\_total\_amount := lv\_total\_amount - lv\_price;

IF lv\_total\_amount >= 0 THEN

lv\_total\_spent := lv\_total\_spent + lv\_price;

lv\_number\_items := lv\_number\_items + 1;

DBMS\_OUTPUT.PUT\_LINE(lv\_total\_amount);

END IF;

EXIT WHEN lv\_total\_amount <= 0;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Total amount spent: ' || lv\_total\_spent);

DBMS\_OUTPUT.PUT\_LINE('Number of items: ' || lv\_number\_items);

END;

ASSIGNMENT 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 flowchart 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 |



Assignment 2-8: Using IF statements

Create a block to accomplish the task outlined in Assignment 2-7. 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

lv\_member\_status VARCHAR2(1) := UPPER('y');

lv\_no\_items\_purchased NUMBER(4) := 11;

lv\_shipping\_cost NUMBER(5,2);

BEGIN

CASE

WHEN lv\_member\_status = 'N' THEN

CASE

WHEN lv\_no\_items\_purchased >=0 AND lv\_no\_items\_purchased <=3 THEN lv\_shipping\_cost := 5.00;

WHEN lv\_no\_items\_purchased >=4 AND lv\_no\_items\_purchased <=6 THEN lv\_shipping\_cost := 7.50;

WHEN lv\_no\_items\_purchased >=7 AND lv\_no\_items\_purchased <=10 THEN lv\_shipping\_cost := 10.00;

ELSE lv\_shipping\_cost := 12.00;

END CASE;

WHEN lv\_member\_status = 'Y' THEN

CASE

WHEN lv\_no\_items\_purchased >=0 AND lv\_no\_items\_purchased <=3 THEN lv\_shipping\_cost := 3.00;

WHEN lv\_no\_items\_purchased >=4 AND lv\_no\_items\_purchased <=6 THEN lv\_shipping\_cost := 5.00;

WHEN lv\_no\_items\_purchased >=7 AND lv\_no\_items\_purchased <=10 THEN lv\_shipping\_cost := 7.00;

ELSE lv\_shipping\_cost := 9.00;

END CASE;

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Member: ' || lv\_member\_status);

DBMS\_OUTPUT.PUT\_LINE('Number of items purchased:' || lv\_no\_items\_purchased);

DBMS\_OUTPUT.PUT\_LINE('Shipping cost: ' || lv\_shipping\_cost);

END;

## HANDS-ON ASSIGNMENTS PART II

ASSIGNMENT 2-9: Using a FOR Loop

Create a PL/SQL block using a 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

lv\_donor\_balance NUMBER(8,2):= 1000.00;

lv\_payment\_due\_date DATE := '01-JAN-13';

lv\_monthly\_payment NUMBER(7,2) := 100.00;

lv\_number\_payments NUMBER(3) := 10;

lv\_get\_month VARCHAR2(10);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Payment Number' || ' Balance ' || 'Payment ' || 'Due Date' );

FOR i IN 1..10 LOOP

lv\_get\_month := EXTRACT(MONTH FROM lv\_payment\_due\_date);

IF lv\_get\_month IN (1,3,5,7,8,10,12) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '31' day;

ELSIF lv\_get\_month IN (4,6,9,11) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '30' day;

ELSE

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '28' day;

END IF;

lv\_donor\_balance := lv\_donor\_balance - lv\_monthly\_payment;

DBMS\_OUTPUT.PUT\_LINE(i || ' ' || lv\_donor\_balance|| ' '|| lv\_monthly\_payment || ' ' || lv\_payment\_due\_date);

END LOOP;

END;

ASSIGNMENT 2-10: Using a Basic Loop

Accomplish the task in Assignment 2-9 by using a basic loop structure.

DECLARE

lv\_donor\_balance NUMBER(8,2):= 1000.00;

lv\_payment\_due\_date DATE := '01-JAN-13';

lv\_monthly\_payment NUMBER(7,2) := 100.00;

lv\_number\_payments NUMBER(3) := 10;

lv\_get\_month VARCHAR2(10);

lv\_cnt\_num NUMBER(2):= 1;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Payment Number' || ' Balance ' || 'Payment ' || 'Due Date' );

LOOP

lv\_get\_month := EXTRACT(MONTH FROM lv\_payment\_due\_date);

IF lv\_get\_month IN (1,3,5,7,8,10,12) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '31' day;

ELSIF lv\_get\_month IN (4,6,9,11) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '30' day;

ELSE

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '28' day;

END IF;

EXIT WHEN lv\_donor\_balance <= 0;

lv\_donor\_balance := lv\_donor\_balance - lv\_monthly\_payment;

DBMS\_OUTPUT.PUT\_LINE(lv\_cnt\_num || ' ' || lv\_donor\_balance|| ' '|| lv\_monthly\_payment || ' ' || lv\_payment\_due\_date);

lv\_cnt\_num := lv\_cnt\_num + 1;

END LOOP;

END;

ASSIGNMENT 2-11: Using a WHILE Loop

Accomplish the task in Assignment 2-9 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

lv\_donor\_balance NUMBER(8,2):= 1000.00;

lv\_amount\_paid NUMBER(8,2) := 0;

lv\_payment\_due\_date DATE := '01-JAN-13';

lv\_monthly\_payment NUMBER(7,2) := 100.00;

lv\_number\_payments NUMBER(3) := 10;

lv\_get\_month VARCHAR2(10);

lv\_cnt\_num NUMBER(2):= 1;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Payment Number' || ' Balance ' || 'Payment ' || 'Due Date' );

WHILE lv\_amount\_paid < lv\_donor\_balance LOOP

lv\_get\_month := EXTRACT(MONTH FROM lv\_payment\_due\_date);

IF lv\_get\_month IN (1,3,5,7,8,10,12) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '31' day;

ELSIF lv\_get\_month IN (4,6,9,11) THEN

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '30' day;

ELSE

lv\_payment\_due\_date := lv\_payment\_due\_date + interval '28' day;

END IF;

lv\_amount\_paid := lv\_amount\_paid + lv\_monthly\_payment;

DBMS\_OUTPUT.PUT\_LINE(lv\_cnt\_num || ' ' || lv\_amount\_paid|| ' '|| lv\_monthly\_payment || ' ' || lv\_payment\_due\_date);

EXIT WHEN lv\_donor\_balance <= 0;

END LOOP;

END;

ASSIGNMENT 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 fo 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%, 2 = 100%, and other = 0. Display the calculated amount.

ASSIGNMENT 2-13: Using nested IF Statements

An organization has committed to matching pledge amounts based on the donor type and pledge amount. Donor types include I = Individual, B = Business organization, and G = Grant funds. The matching percent’s are to be applied as follows:

|  |  |  |
| --- | --- | --- |
| Donor Type | Pledge Amount | Matching % |
| I | $100-$249 | 50% |
| I | $250-$499 | 30% |
| I | %500 or more | 20% |
| B | $100-$499 | 20% |
| B | $500-$999 | 10% |
| B | $1,000 or more | 5% |
| G | $100 or more | 5% |

Create a PL/SQL block using nested IF statements to accomplish the task. Input values for the block are the donor type code and the pledge amount.

DECLARE

lv\_donor\_type CHAR(1) NOT NULL := upper('G');

lv\_pledge\_amount NUMBER(5) NOT NULL :=1000;

lv\_matching\_percent NUMBER (4,2) NOT NULL :=0;

BEGIN

IF lv\_donor\_type = 'I' THEN

IF lv\_pledge\_amount >= 100 AND lv\_pledge\_amount <= 249 THEN

lv\_matching\_percent := 0.50;

ELSIF lv\_pledge\_amount >= 250 AND lv\_pledge\_amount <= 499 THEN

lv\_matching\_percent := 0.30;

ELSIF lv\_pledge\_amount >= 500 THEN

lv\_matching\_percent := 0.20;

END IF;

ELSIF lv\_donor\_type = 'B' THEN

IF lv\_pledge\_amount >= 100 AND lv\_pledge\_amount <= 499 THEN

lv\_matching\_percent := 0.20;

ELSIF lv\_pledge\_amount >= 500 AND lv\_pledge\_amount <= 999 THEN

lv\_matching\_percent := 0.10;

ELSIF lv\_pledge\_amount >= 1000 THEN

lv\_matching\_percent := 0.05;

END IF;

ELSIF lv\_donor\_type = 'G' THEN

IF lv\_pledge\_amount >= 100 THEN

lv\_matching\_percent := 0.05;

END IF;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Donor type not recognized');

END IF;

DBMS\_OUTPUT.PUT\_LINE('Donor Type: ' || lv\_donor\_type);

DBMS\_OUTPUT.PUT\_LINE('Pledge Amount: ' || lv\_pledge\_amount);

DBMS\_OUTPUT.PUT\_LINE('Matching %: ' || lv\_matching\_percent \* 100 || '%');

END;

## CASE PROJECTS

Case 2-1: Flowcharting

Find a Web site with basic information on flowcharting. Describe at least two interesting aspects of flowcharting discussed on the Web site.

Case 2-2: Working with More Movie Rentals

The More Movie Rentals Company wants to display a rating value for a movie based on the number of times the movie has been rented. The rating assignments are outlined in the following chart.

|  |  |
| --- | --- |
| Number of Rentals | Rental Rating |
| Up to 5 | Dump |
| 5-20 | Low |
| 21-35 | Mild |
| More than 35 | High |

Create a flowchart and then a PL/SQL block to address the processing needed. The block should determine and then display the correct rental rating. Test the block, using a variety of rental amounts.