# In-Class Practice – (PL/SQL)

GOALS: Spend ½ your time on #1-8, then plan to spend rest of time on 9-10. You can jump around. We'll post the solution.

1. **Practice running PL/SQL**
   a. Run Query A portion to know the balance due for vendor 95.
   b. Then run the PL/SQL code that calculates the same thing using PL/SQL. If you don't get an output discuss why and how to fix that with in your breakout room.
   c. Run the proper command to correct the output issue.
   d. Rerun the process of Query A and B but this time for vendor_id 34 and discuss results.

```
-- Query A
SELECT SUM(invoice_total - payment_total - credit_total) balance_due
FROM invoices
WHERE vendor_id = 95;

-- Query B
--CONNECT ap/ap;  --don't need to connect since we are already connected to our database
--SET SERVEROUTPUT ON;

DECLARE
  sum_balance_due_var NUMBER(9, 2);
BEGIN
  SELECT SUM(invoice_total - payment_total - credit_total) balance_due
  INTO sum_balance_due_var
  FROM invoices
  WHERE vendor_id = 95;
  IF sum_balance_due_var > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Balance due: $' ||
                         ROUND(sum_balance_due_var, 2));
  ELSE
    DBMS_OUTPUT.PUT_LINE('Balance paid in full');
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred');
END;
/
```

2. **Understanding outputs using PUT_LINE function –** Write a piece of PL/SQL code that uses the PUT_LINE() function to output the phrase that looks something like this: 'Hello, my name is [INSERT YOUR NAME].'.

3. **Using Variables**
   a. Run the following code and note how it is define variables in different ways. Note that PL/SQL uses the := operator and not the usual = or == operator.

```
DECLARE
  max_invoice_total  invoices.invoice_total%TYPE;
  min_invoice_total  invoices.invoice_total%TYPE;
  percent_difference NUMBER;
  count_invoice_id   NUMBER;
  vendor_id_var      NUMBER := 95;
BEGIN
  SELECT MAX(invoice_total), MIN(invoice_total), COUNT(invoice_id)
  INTO max_invoice_total, min_invoice_total, count_invoice_id
  FROM invoices WHERE vendor_id = vendor_id_var;
  percent_difference :=
      (max_invoice_total - min_invoice_total) / min_invoice_total * 100;

  DBMS_OUTPUT.PUT_LINE('Maximum invoice: $' || max_invoice_total);
  DBMS_OUTPUT.PUT_LINE('Minimum invoice: $' || min_invoice_total);
  DBMS_OUTPUT.PUT_LINE('Percent difference: %' || ROUND(percent_difference, 2));
  DBMS_OUTPUT.PUT_LINE('Number of invoices: ' || count_invoice_id);
END;
/
```

   b. After you are familiar with variables, go make a copy of the code from #2 and add more code that *declares* a variable called MyAge with a default value of your age and then outputs that in a separate line that looks like 'I am 39 years old'.

4. **Substitution Variables:** Update the code you wrote for #3 but this time instead of hardcoding the value, default the value of the MyAge variable to be equal to a substitution variable called user_defined_age. NOTE: If you get an error on this, run the following command to allow user-defined variables: `SET DEFINE ON;`
    a. Remember that substitution variables are define by preceding the variable with a '&' symbol.
    b. After you've created a new version of the code, run it and see if you are prompted to enter an age and if the output still works.

5. **SELECT a value INTO a variable and use it for Arithmetic and Logic**
    a. Create a PL/SQL statement that will select the following SQL statements below into two different variables. Use the variables to calculate the percentage of vendors whose phone numbers need to be updated.
    b. Hint: Code incrementally (i.e. write a little and run. Then write a little and run. Etc….)
        i. Start by declaring two variables to store vendors who have phones and those missing phones.
        ii. Then select the values from each SQL statement into the variables. After that output the values.
        iii. Once you have the values outputting, you can concatenate the strings to make the first two lines
        iv. When you get to the 3rd line, create the arithmetic to calc the percent and then output it

| **Starter SQL statements** | **Output Goal** |
|---|---|
| `select count(distinct vendor_id) as not_missing`<br>`from vendors`<br>`where vendor_phone is not null;`<br><br>`select count(distinct vendor_id) as missing`<br>`from vendors`<br>`where vendor_phone is null;` | 97 vendors have a phone on file<br>25 vendors do not have a phone on file<br>20.49% of our vendors need updated phone numbers<br><br>*Hint on line 3: Use **missing / (missing + not_missing) * 100** to get the output of line 3.* |

**6. IF statement or CASE to handle logical flow**
    a. If the percentage of missing phones is greater than 10%, output a forth line that says, 'ALERT: Time to update vendor phone'
    b. If the percentage of missing phones is 10% or less, output a different version of the forth line that says 'Vendor phone: no need to update'

**7. Loop**
    a. Write a FOR loop that outputs the blue text below
    *b.* One you get this to work, update the code to only output even numbers (*hint: if mod(i,2)= 0*)

```
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
```

8. **Dynamic SQL**
    a. Run the sample code and familiarize yourself with how it works and what it's doing.
    b. After you're familiar, update the code to do the following:
        i. Prompt the user for a variable called terms_id and another called invoice_id. Then replace the values of these variables in the SQL string.
        ii. Test that your code works (i.e. captures input from the user and concatenates it in the SQL)
        iii. Comment out the code that prints the SQL and add in a line that will EXECUTE IMMEDIATE the dynamic_sql string.

```
DECLARE
  dynamic_sql VARCHAR2(400);
BEGIN

  dynamic_sql := 'UPDATE invoices ' ||
                 'SET terms_id = ' || 1 || ' ' ||
                 'WHERE invoice_id = ' || 1;

  DBMS_OUTPUT.PUT_LINE(dynamic_sql);

END;
/
```

9. **Bulk Collect**
    a. Run the sample code and familiarize yourself with how it works and what it's doing.
    b. After you're familiar, update the code to do the following:
        i. Declare a type of payments_table that is numeric
        ii. Define a payments_table called payment_amounts
        iii. Select payment_total from invoices into the payment_amounts variable for the first 20 rows. Continue to sort by vendor_id.
        iv. Update the FOR Loops to loop through payment_amounts instead of vendor_names and output something like the following.
            Payment amount 1: 117
            Payment amount 2: 1084
        v. If you want to practice your IF statements, only output the value if it's not zero.

```
DECLARE
  TYPE names_table IS TABLE OF VARCHAR2(40);
  vendor_names   names_table;     --bulk collect will dynamic set the length of your list/array as it is collected
BEGIN
  SELECT vendor_name
  BULK COLLECT INTO vendor_names
  FROM vendors
  WHERE rownum <= 10 ORDER BY vendor_id
  ;

  FOR i IN 1..vendor_names.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Vendor name ' || i || ': ' ||
            vendor_names(i));
  END LOOP;
END;
/
```

**10. Cursor**

a. Run the sample code and familiarize yourself with how it works and what it is doing.
b. After you're familiar, try to create a new PL/SQL statement that does the following:
   i. Declare a cursor called vendor_cursor based on the following SQL
   ```
   SELECT vendor_id, vendor_name, vendor_state, vendor_phone
   FROM vendors
   where vendor_phone is null;
   ```
   ii. Update the row variable's name and set it to be based on the rowtype of vendors
   iii. Update the loop to loop through the vendor_cursor
   iv. If the vendor_state = 'CA' or if vendor_state = 'OH', update the vendor_phone to the value of 'Update Immediately'.
   v. NOTE: don't focus on why the statement is being run. Just figure out how to make it work

```
DECLARE
    CURSOR invoices_cursor IS
        SELECT invoice_id, invoice_total
        FROM invoices
        WHERE invoice_total - payment_total - credit_total > 0;

    invoice_row invoices%ROWTYPE;

BEGIN
    FOR invoice_row IN invoices_cursor LOOP

        IF (invoice_row.invoice_total > 5000) THEN
            UPDATE invoices
            SET credit_total = credit_total - (invoice_total * .05)
            WHERE invoice_id = invoice_row.invoice_id;

            DBMS_OUTPUT.PUT_LINE('Credit_total increased by 10% ($' ||
                ROUND((invoice_row.invoice_total *.05),2) ||
                ') for invoice ' || invoice_row.invoice_id);
        ELSIF (invoice_row.invoice_total > 1000) THEN
            UPDATE invoices
            SET credit_total = credit_total - (invoice_total * .1)
            WHERE invoice_id = invoice_row.invoice_id;

            DBMS_OUTPUT.PUT_LINE('Credit_total increased by  5% ($' ||
ROUND((invoice_row.invoice_total *.1),2) || ') for invoice ' || invoice_row.invoice_id);

        END IF;
    END LOOP;

END;
/
```

## Extra practice

11. **Handling errors** – Make an update to the EXCEPTION block of the code below to better handle the error that is occurring. You can try this in two ways. (e.g. WHEN OTHERS to capture general errors or uses pre-defined errors). Hint the error is due to trying to enter a duplicate primary key.

```
BEGIN
    insert into terms VALUES (1,'Net due 10', 10);
    DBMS_OUTPUT.PUT_LINE('1 row inserted.');
EXCEPTION

END;
/
```