# In-Class Practice – Day 17 (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 solution.

1. **Practice running PL/SQL**
   1. Run Query A portion to know the balance due for vendor 95.
   2. 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.
   3. Run the proper command to correct the output issue.
   4. 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;

/

1. **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].’.
2. **Using Variables**
   1. 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 = operator.
   2. 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’.

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;

/

1. **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;
   1. Remember that substitution variables are define by preceding the variable with a ‘&’ symbol.
   2. 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.
2. **SELECT a value INTO a variable and use it for Arithmetic and Logic**
   1. 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.
   2. Hint: Code incrementally (i.e. write a little and run. Then write a little and run. Etc….)
      1. Start by declaring two variables to store vendors who have phones and those missing phones.
      2. Then select the values from each SQL statement into the variables. After that output the values.
      3. Once you have the values outputting, you can concatenate the strings to make the first two lines
      4. 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  from vendors where vendor\_phone is not null;  select count(distinct vendor\_id) as missing  from vendors where vendor\_phone is null; | 97 vendors have a phone on file  25 vendors do not have a phone on file  20.49% of our vendors need updated phone numbers  *Hint on line 3: Use* ***missing / (missing + not\_missing) \* 100*** *to get the output of line 3.* |

1. **IF statement or CASE to handle logical flow**
   1. If the percentage of missing phones is greater than 10%, output a forth line that says, ‘ALERT: Time to update vendor phone’
   2. 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’
2. **Loop**
   1. Write a FOR loop that outputs the blue text below
   2. 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

1. **Dynamic SQL**
   1. Run the sample code and familiarize yourself with how it works and what it’s doing.
   2. After you’re familiar, update the code to do the following:
      1. 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.
      2. Test that your code works (i.e. captures input from the user and concatenates it in the SQL)
      3. 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;

/

1. **Bulk Collect**
   1. Run the sample code and familiarize yourself with how it works and what it’s doing.
   2. After you’re familiar, update the code to do the following:
      1. Declare a type of payments\_table that is numeric
      2. Define a payments\_table called payment\_amounts
      3. Select payment\_total from invoices into the payment\_amounts variable for the first 20 rows. Continue to sort by vendor\_id.
      4. 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

* + 1. 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;**

**/**

1. **Cursor**
   1. Run the sample code and familiarize yourself with how it works and what it is doing.
   2. After you’re familiar, try to create a new PL/SQL statement that does the following:
      1. 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;

* + 1. Update the row variable’s name and set it to be based on the rowtype of vendors
    2. Update the loop to loop through the vendor\_cursor
    3. If the vendor\_state = ‘CA’ or if vendor\_state = ‘OH’, update the vendor\_phone to the value of ‘Update Immediately’.
    4. 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**

1. **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;**

**/**