Practical 3: PL/SQL

Learning Objectives:

1. Basic PL/SQL block structure
2. Variable, constant declaration
3. Control Statements – Conditional and iterative statements
4. Stored Procedures
5. Exceptions
6. Functions and Local Procedures

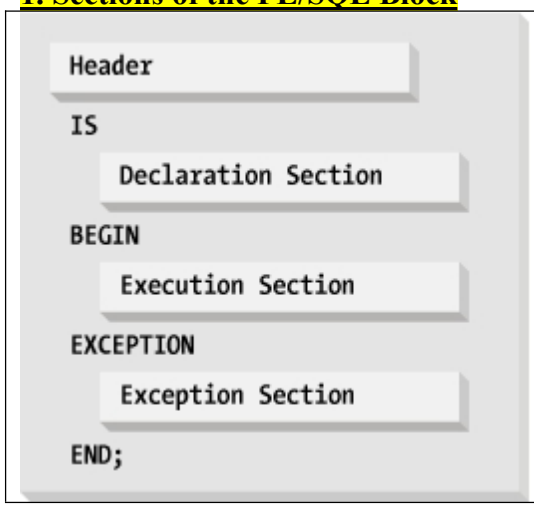
References

1. Oracle PL/SQL Language Reference
   https://docs.oracle.com/en/database/oracle/oracle-database/18/lnpls/database-pl-sql-language-reference.pdf

2. https://docs.oracle.com/en/database/oracle/oracle-database/19/tdddg/

PL/SQL stands for "Procedural Language extensions to the Structured Query Language. Oracle Corporation introduced PL/SQL to overcome some limitations in SQL and to provide a more complete programming solution for those who sought to build mission critical applications to run against the Oracle database.

1. Sections of the PL/SQL Block



Header
Used only for named blocks. Optional.

Declaration section
Identifies variables, cursors, and sub-blocks that are referenced in the execution and exception sections. Optional.

Execution section
Statements the PL/SQL runtime engine will execute at runtime. Mandatory.

Exception section
Handles exceptions to normal processing (warnings and error conditions). Optional.

You can write a PL/SQL code as an ***anonymous block***, ***a procedure*** or ***a function***.
An anonymous block is generally used by developers to test their coding. It functions as scripts that execute PL/SQL statements, usually including calls to procedures and functions.

2. Variable and Constant declaration

Variables must begin with a letter and may be up to 30 characters long.
PL/SQL is not case-sensitive.
The PL/SQL data types include the SQL data types.

```
DECLARE
        wages NUMBER(7,2);
        hours_worked NUMBER not null:= 40;
        hourly_salary NUMBER := 22.50;
        country VARCHAR2(128);
        counter NUMBER := 0;
        done BOOLEAN;
        valid_id BOOLEAN;
         postcode CHAR(5);
         emp_dob DATE;

        cust_fname  customers.contactFirstName%TYPE;
        cust_rec1 customers%ROWTYPE;
        cust_rec2 customers%ROWTYPE;

        MAX_PROPERTIES CONSTANT NUMBER:=30;

        TYPE commissions IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
        comm_tab commissions;
BEGIN
        wages := (hours_worked * hourly_salary);
        country := 'France';
        country := UPPER('Canada');
        done := (counter > 100);
        valid_id := TRUE;
        comm_tab(5) := 20000 * 0.15;


        DBMS_OUTPUT.PUT_LINE('array value comm_tabl(5) : '|| comm_tab(5));

        DBMS_OUTPUT.put_line (
                CASE
                        WHEN valid_id THEN 'You can proceed'
                        WHEN NOT valid_id THEN 'No access allowed'
                END
        );
END;
/
```

3. Control Statements

Conditional statements

- IF..END IF
- CASE

**IF condition THEN
statements
END IF;**
	**IF condition THEN
	statements
	ELSE
	else_statements
	END IF;**
		**IF condition_1 THEN
		statements_1
		ELSIF condition_2 THEN
		statements_2
		[ELSIF condition_3 THEN
		statements_3
		]...
		[ELSE
		else_statements
		]
		END IF;**

CASE selector WHEN selector_value_1 THEN statements_1 WHEN selector_value_2 THEN statements_2 ... WHEN selector_value_n THEN statements_n [ELSE else_statements] END CASE;]	CASE WHEN condition_1 THEN statements_1 WHEN condition_2 THEN statements_2 ... WHEN condition_n THEN statements_n [ELSE else_statements] END CASE;]

Example

```
1. DECLARE
2.    v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
3.    v_day VARCHAR2(15);
4. BEGIN
5.    v_day := RTRIM(TO_CHAR(v_date, 'DAY'));
6.    IF v_day IN ('SATURDAY', 'SUNDAY') THEN
            DBMS_OUTPUT.PUT_LINE (v_date||' falls on weekend');
7.    END IF;
8.    DBMS_OUTPUT.PUT_LINE ('Done...');
9. END;
```

Repetitive statements

- LOOP..END LOOP
- WHILE loop
- FOR loop

[label] LOOP statements END LOOP [label];	[label] FOR index IN [REVERSE] lower_bound..upper_bound LOOP statements END LOOP [label];	[label] WHILE condition LOOP statements END LOOP [label];

LOOP
x := x + 1;
IF x > 3 THEN
EXIT;
END IF;
END LOOP;
 	
LOOP
...
x := x + 1;
EXIT WHEN x > 3;
END LOOP
	SELECT COUNT(employee_id)
	INTO emp_count
	FROM employees;
	
	FOR i IN 1..emp_count LOOP
	INSERT INTO temp
	VALUES(i, 'to be added later');
	END LOOP;
		done BOOLEAN := FALSE;
		
		BEGIN
		WHILE done LOOP
		....
		....
		END LOOP;

4. Stored Procedures

5

The **Proc1.txt** contains a procedure ***PRC_PRICE_INCR*** that receives an input parameter ***IN_Product_Code*** and increases the MSRP by 10% and writes a record into a ***Price_Audit*** table.

```
CREATE OR REPLACE PROCEDURE PRC_PRICE_INCR(IN_Product_Code IN
VARCHAR2) IS
        v_productname varchar2(70);
        v_oldprice number(7,2);
        v_newprice number(7,2);
BEGIN
-- GET THE PRODUCT PRICE
      SELECT ProductName, MSRP INTO v_productname, v_oldprice
        FROM PRODUCTS
        WHERE ProductCode = IN_Product_Code;

      v_newprice := round(v_oldprice *1.1,2);

-- UPDATE the record
      UPDATE PRODUCTS
          SET MSRP = v_newprice
        WHERE ProductCode = IN_Product_Code;

-- WRITE DATA TO THE PRICE_AUDIT TABLE
      INSERT INTO Price_Audit VALUES(IN_Product_Code, v_productname,
            v_oldprice, v_newprice, USER, SYSDATE);

      DBMS_OUTPUT.PUT_LINE('Product '||IN_Product_Code||' '||v_productname||
            ' new price is $' ||v_newprice);
END;
```

The structure of ***Price_Audit*** table is as follows:

```
create table Price_Audit
(  IN_Product_Id      varchar(15),
   v_productname      varchar(70),
   v_oldprice         number(7,2),
   v_newprice         number(7,2),
   user_name          varchar(20),
   date_change        date
);
```

a. Compile and run the ***PRC_PRICE_INCR*** procedure:
b. Check the data in the ***Price_Audit*** table
c. Explain why a stored procedure is better compared to writing DQL and DML at the command line.

If you would like to get a view about the parameters of the procedure, just use
SQL> desc <procedure name>;

If you would like to see the code for the procedure, then use (assuming you are logged in as the owner of the procedure)

SQL> SELECT Text FROM User_Source WHERE Name ='PROCEDURENAME' ORDER BY Line;

Note:
The DBMS_OUTPUT.PUT_LINE procedure writes the message to be displayed to the buffer for storage. Once a program has been completed, the information in the buffer is displayed on the screen. The size of the buffer is initialized at 2000 bytes, and can be set between 2000 and 1,000,000 bytes. To change the buffer size, use the following command

set serveroutput on size 1000000

6. Exceptions

```
create or replace procedure prc_customer_purchase(cust_num in number) is
  v_purchase   varchar2(3):='NO';
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Check if the customer has made any purchases');
  SELECT 'YES'
    INTO v_purchase
    FROM orders
   WHERE customernumber = cust_num;

   DBMS_OUTPUT.PUT_LINE ('The customer has only made 1 purchase.');
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
    DBMS_OUTPUT.PUT_LINE ('The customer has not made any purchases');

  WHEN TOO_MANY_ROWS
  THEN
     DBMS_OUTPUT.PUT_LINE ('The customer has made more than 1 purchases.');
END;
```

NB. When issuing any SQL command such as SELECT, INSERT, UPDATE and DELETE, there may not be any rows returned. This (zero rows) of a SELECT..INTO will cause an exception and must be checked by an exception handler.

User-Defined Exception


```
create or replace procedure prc_check_purchase(cust_no in number) is
  v_total_purchases NUMBER;
  e_invalid_no    EXCEPTION;
BEGIN
  IF cust_no < 0
  THEN
    RAISE e_invalid_no;
  END IF;

  SELECT COUNT(*)
   INTO v_total_purchases
   FROM orders
   WHERE customernumber = cust_no;

  DBMS_OUTPUT.PUT_LINE ('The customer has made '||v_total_purchases||'  purchases');
  DBMS_OUTPUT.PUT_LINE ('No exception has been raised');
EXCEPTION
  WHEN e_invalid_no then
    DBMS_OUTPUT.PUT_LINE ('Customer number cannot be negative');
END;
```

Refer Chapter 11 page 11-10 for a list of predefined exceptions.

RAISE_APPLICATION_ERROR is a special built-in procedure provided by Oracle.

RAISE_APPLICATION_ERROR(error_number, error_message, [keep_errors]);

It allows programmers to create meaningful error messages for a specific application, and it works with user-defined exceptions.

error_number is a number that a programmer associates with a specific error message. It can be any number between –20,999 and –20,000.

error_message is the text of the error, can have up to 2,048 characters.

keep_errors is a Boolean parameter. If keep_errors is set to TRUE, the new error is added to the list of errors that have been raised already.

[Refer to PL SQL Guide Page 11-18]

```
CREATE PROCEDURE account_status (  due_date DATE,  today DATE) IS
BEGIN
  IF due_date < today THEN -- explicitly raise exception
     RAISE_APPLICATION_ERROR(-20000, 'Account past due.');
  END IF;
END;
/
DECLARE
  past_due EXCEPTION; -- declare exception
  PRAGMA EXCEPTION_INIT (past_due, -20000); -- assign error code to exception
```

```
BEIN
  account_status (TO_DATE('01-JUL-2020', 'DD-MON-YYYY'),
  TO_DATE('09-JUL-2020', 'DD-MON-YYYY')); -- invoke procedure
EXCEPTION
  WHEN past_due THEN -- handle exception
   DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQLERRM(-20000)));
END;
/
```

EXCEPTION_INIT pragma

A pragma is a special instruction to the PL/SQL compiler. The EXCEPTION_INIT pragma allows you to associate an Oracle error number with the name of a user-defined error; after which you can reference the error and write a handler for it.

```
exception_name EXCEPTION;
PRAGMA EXCEPTION_INIT(exception_name, error_code);
```

6. Functions and Local Procedures

```
Create or replace function NumberOrders (v_custNo IN number)
        return number as v_count NUMBER;
begin
    select count(*)
        into v_count
        from orders
        where customernumber = v_custno;
    return v_count;
end;
```

```
declare
  cEmployee_ID  employees.employeenumber%type;
  cFirstName    employees.FirstName%type;
  status        boolean;
  cust_num      orders.customernumber%type;

 procedure GetEmployeeDetails (
  EmplNo     in     employees.Employeenumber%type,
  EmpName    out    employees.FirstName%type,
  status     out    Boolean) is
 begin
    select FirstName
       into EmpName   from Employees E
       where E.Employeenumber= EmplNo;
```

```
    status :=true;

    exception
      when no_data_found then
         status :=false;
  end;

begin
   cEmployee_ID:= '1611';
   GetEmployeeDetails(cEmployee_ID,cFirstName,status);
   if (status) then
       dbms_output.put_line(cEmployee_ID||'  '||cFirstName);
   else
       dbms_output.put_line (' Employee '||cEmployee_ID||'  not  found');
   end if;

    cust_num := 496;
    dbms_output.put_line ('Number of orders by customer '||cust_num||
                    ' is '|| NumberOrders(cust_num) );
end;
/
```

Exercise

1. Write a procedure that will receive ***CustomerNumber*** as input and print a message to show how long ago (the duration) the last order was made by this customer.
   e.g.
   Customer [xxxxx] last made an order on [dd/mm/yyyy].
   That was [xxx] months ago. <---show this message if the duration is 30 days or more.

   That was [xxx] days ago. <---show this message if the duration is less than 30 days.

2. In the retail business, when a customer makes an order, the **sale price of an item** may be lower than the indicated MSRP (Manufacturer Selling Recommended Price). In order to ensure that a profit is made, the sale price cannot be lower than (105% x buyPrice)

   A customer had noticed that you had not given him a 5% discount that he is entitled to for a product. Your company policy is customers can make a claim within 30 days from the order date.

   Write a procedure to update the order details (if the claim is valid) or display an

appropriate message why the claim is rejected. The input parameters are ***OrderNo*** and ***ProductCode***.

```
For testing,

select ordernumber, orderdate from orders;

select p.productcode, buyprice, priceeach
  from products p, orderdetails o
  where p.productcode = o.productcode and
      ordernumber = 10425;
```

3. Rewrite the ***PRC_PRICE_INCR*** procedure to display an error message when a ProductCode that does not exist has been entered.