PROGRAM 1

Write a PL/SQL block to calculate the incentive of an employee whose ID is 110.

```
DECLARE
 v_employee_id
employees.employee_id %TYPE := 110;
 v_salary   employees.salary %TYPE;
 v_incentive   NUMBER(10,2);
 c_incentive_rate  CONSTANT NUMBER:=0.10;

BEGIN

  Select salary
  into v_salary
  FROM employees
  where employee_id = v_employee_id;
 v_incentive := v_salary * c_incentive_rate;
DBMS_OUTPUT.PUT_LINE('Employee ID:' || v_employee_id);
DBMS_OUTPUT.PUT_LINE('Salary : $' || TO_CHAR
(v_salary, '99,999.00'));
DBMS_OUTPUT.PUT_LINE('Calculated Incentive : $' || TO_CHAR
(v_incentive, '9999.00'));
UPDATE employees SET incentive = v_incentive where
employee_id = v_employee_id;
 EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Employee with
ID' || v_employee_id || 'not found!');
 When others Then
    DBMS_OUTPUT.PUT_LINE('An unexpected error
occurred:' || SQLERRM);
 END;
```

PROGRAM 2

Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

```
DECLARE:
"eMpLoYeE_NaMe" VARCHAR2(50):='Alice';
V-name VARCHAR2(50);
BEGIN
V-name := "eMpLOYeE_NaMe";
DBMS-OUTPUT.PUT-LINE ('Valid reference value: ' || V-name);
DBMS-OUTPUT.PUT-LINE ("EMPLOYEE_NAME" ||
eMpLoYeE_NaMe");
END;
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122

Sample table: employees

```
DECLARE
v_employee_id CONSTANT employees.employee_id %TYPE:=122;
C_adjustment_rate CONSTANT NUMBER:=0.10;
V_old_salary employees.salary %TYPE;

BEGIN
  select salary
  INTO V_old_salary
  FROM employees
  where employee_id = V_employee_id;
IF SQL %ROWCOUNT >0 THEN
  DBMS_OUTPUT.PUT_LINE ('Employee ID' || v_employee_id ||
  'salary adjusted successfully.');
  DBMS_OUTPUT.PUT_LINE ('old salary: $' ||
TO_CHAR(V_old_salary, '99999.00'));
  COMMIT;
ELSE
  DBMS_OUTPUT.PUT_LINE ('Error: No employee
found with ID' || v_employee_id || '. No update performed.');
ENDIF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('Error: Employee with ID' ||
V_employee_id || 'not found in the initial check!');
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('An unexpected error occured:'
      || SQLERRM);
      ROLLBACK;
  END;
```

PROGRAM 4

Write a PL/SQL block to create a procedure using the "IS [NOT] NULL Operator" and show AND operator returns TRUE if and only if both operands are TRUE.

```
CREATE OR REPLACE PROCEDURE DEMO-LOGIC (P_input_val
IN VARCHAR2) IS V-operand_a Boolean := TRUE, V-operand_b
BOOLEAN:=TRUE; V-and_result BOOLEAN; BEGIN
DBMS_OUTPUT.PUT_LINE('IS [NOT] NULL Demonstration');
IF P_input_val IS NULL THEN DBMS_OUTPUT.PUT_LINE
('Input value is NULL. (P_input_val IS NULL is TRUE)');
ELSE
   DBMS_OUTPUT.PUT_LINE ('Input value is NOT NULL: " || P_input_val ||
   "");
   ENDIF;
IF (P-input_val IS NOT NULL THEN
   DBMS-OUTPUT.PUT-LINE ('P-input_val IS NOT NULL is TRUE');
   ELSE
   DBMS-OUTPUT.PUT-LINE ('(P-input_val IS NOT NULL is FALSE)');
   END IF;
      V-operand_a := TRUE; V-operand_b := TRUE; V-and_result :=
~~DBMS_OUTPUT.PUT_LINE (CHR(10)||~~
V-operand_a AND V-operand_b;
IF V-and_result THEN
   DBMS_OUTPUT.PUT_LINE ('TRUE AND TRUE = TRUE (Result: TRUE)');
END IF;
V-operand_a := FALSE; V-operand_b := TRUE; V-and_result := V-operand_a
AND V-operand_b;
IF NOT V-and_result THEN
   DBMS_OUTPUT.PUT_LINE ('FALSE AND TRUE = FALSE
   (Result: FALSE)');
END IF;
END;
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
DECLARE
   v_test_string VARCHAR2(50) := 'Sales_Q1_25%';
   v_message VARCHAR2(100);
BEGIN
    IF v_test_string LIKE 'Sales%' THEN
        DBMS_OUTPUT.PUT_LINE('1. Matched "Sales%":
 || v_test_string || 'Starts with "Sales".');
  ENDIF;
IF v_test_string LIKE 'Sale_Q1_25%'
THEN
    DBMS_OUTPUT.PUT_LINE('2. Matched "Sale_Q1_25%":
   Has any single character after "Sale".');
   ENDIF;
IF v_test_string LIKE '%25%' THEN
   DBMS_OUTPUT.PUT_LINE('3. Matched "%25%": Contains "25"
   anywhere!');
ENDIF;
IF v_test_string LIKE 'Sales\_Q1\_25\%' ESCAPE '\' THEN
  v_message := '4. Matched (with ESCAPE): The string matches the literal
  pattern "Sales_Q1_25%".';
ELSE
    v_message := '4. Failed to match without ESCAPE.';
ENDIF;
  DBMS_OUTPUT.PUT_LINE('Test string: "' || v_test_string || '"');
  DBMS_OUTPUT.PUT_LINE('Pattern used: "Sales\_Q1\_25\%"
  ESCAPE "\"');
  DBMS_OUTPUT.PUT_LINE(v_message);
END;
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.

```
DECLARE
    num_small NUMBER:=80;
    num_large NUMBER:=35;
    v_temp NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('--Initial Values--');
    DBMS_OUTPUT.PUT_LINE('num_small:' || num_small);
    DBMS_OUTPUT.PUT_LINE('num_large:' || num_large);
IF num_small > num_large THEN
    DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Swap is required...');
    v_temp := num_small;
    num_small := num_large;
    num_large := v_temp;
ELSE
    DBMS_OUTPUT.PUT_LINE(CHR(10) || 'The numbers are already
in the correct order. No swap required.');
END IF;
DBMS_OUTPUT.PUT_LINE(CHR(10)||'---Final Arranged Values--
DBMS_OUTPUT.PUT_LINE('The small number is now in num_
small:' || num_small);
DBMS_OUTPUT.PUT_LINE('The large number is now in
num_large:' || num_large);
END;
```

```
IF SQL %ROWCOUNT=1 THEN
    DBMS_OUTPUT.PUT_LINE('INFO: Record for
Employee '||p_employee_id||' updated. Target NOT met. Incentive set to
$0');
  END IF;
  COMMIT;
```

PROGRAM 7

Write a PL/SQL procedure to calculate the incentive on a target achieved and display the message either the record updated or not.

```
CREATE OR REPLACE PROCEDURE CALC_TARGET_INCENTIVE
p_employee_id IN employees.employee_id %TYPE) IS
c_bonus_amount CONSTANT NUMBER := 500;
v_target_met BOOLEAN := FALSE;
v_actual_sales employees.actual_sales %TYPE;
v_target_sales employees.target_sales %TYPE;

BEGIN
  Select actual_sales, target_sales INTO v_actual_sales,
  v_target_sales FROM employees where employee_id
  = p_employee_id;
IF v_actual_sales >= v_target_sales THEN
      v_target_met := TRUE;
    END IF;
IF v_target_met THEN
      SET incentive = c_bonus_amount where employee_id=
p_employee_id;
 IF SQL %ROWCOUNT=1 THEN
  DBMS_OUTPUT.PUT_LINE('SUCCESS: Record for Employee'||
p_employee_id||'updated. Target met. Incentive set to $'||c_bonus_
amount||'.');
  ELSE
  DBMS_OUTPUT.PUT_LINE('ERROR: Updated failed for Employee
||p_employee_id||'even though target was fetched.');
  END IF;
  ELSE
    Update employees
     SET incentive=0
  where employee_id = p_employee_id;
```

```
ELSE
  DBMS_OUTPUT.PUT_LINE('ERROR: Employee ID '||p_employee_id
'not found. No update performed.');
  ROLLBACK;
END IF;
```

PROGRAM 8

Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

```
CREATE OR REPLACE PROCEDURE CALC_TIERED_
INCENTIVE (p_employee_id IN employees.employee_id%TYPE,
p_actual_sales IN employees.actual_sales%TYPE) IS
v_incentive_rate NUMBER(5,4);
v_calculated_incentive NUMBER(10,2);

BEGIN

  IF p_actual_sales > 100000 THEN
      v_incentive_rate := 0.05;
  ELSIF p_actual_sales >= 50000 THEN
      v_incentive_rate := 0.03;

  ELSE:
      v_incentive_rate := 0.01;
  END IF;
v_calculated_incentive := p_actual_sales * v_incentive_rate;
UPDATE employees
SET incentive = v_calculated_incentive, actual_sales = p_actual_
sales where employee_id = p_employee_id;
IF SQL%ROWCOUNT = 1 THEN
  DBMS_OUTPUT.PUT_LINE('SUCCESS: Employee'||p_
employee_id||' has actual sales of $'||TO_CHAR(p_actual_sales,
'FM99,999.00')||'.');
DBMS_OUTPUT.PUT_LINE('Incentive Rate: '||v_incentive_rate*100
||'%');
DBMS_OUTPUT.PUT_LINE('Incentive Calculated and Updated: $'||
TO_CHAR(v_calculated_incentive,'FM99,999.00')||'.');
COMMIT;
```

```
ID.  ELSE
        DBMS_OUTPUT.PUT_LINE('It is currently
        exactly full.');
        END IF;
    END IF;
END;
```

PROGRAM 9

Write a PL/SQL program to count number of employees in department 50 and check whether this department have any vacancies or not. There are 45 vacancies in this department.

```
DECLARE
  c_dept_id        CONSTANT NUMBER := 50;
  c_total_capacity CONSTANT NUMBER := 45;
  v_employee_count NUMBER;
  v_vacancies NUMBER;
BEGIN
    select COUNT(employee_id)
    INTO v_employee_count
    FROM employees
    where department_id = c_dept_id;
    v_vacancies := c_total_capacity - v_employee_count;
DBMS_OUTPUT.PUT_LINE ('Department' || c_dept_id ||
'Vacancy check');
DBMS_OUTPUT.PUT_LINE('Total slots (Capacity): ' ||
c_total_capacity);
DBMS_OUTPUT.PUT_LINE ('Current Employees :' ||
v_employee_count);
IF  v_vacancies > 0  THEN
    DBMS_OUTPUT.PUT_LINE ('Result: YES, this department
HAS vacancies.');
    DBMS_OUTPUT.PUT_LINE ('Number of vacancies: " ||
    v_vacancies);
ELSIF  v_vacancies = 0  THEN
  DBMS_OUTPUT.PUT_LINE ('Result : No this department has no
                vacancies. It is full.');
```

```
9. ELSE
    DBMS_OUTPUT.PUT_LINE('Result: The department is OVER
    capacity!');
    DBMS_OUTPUT.PUT_LINE('Over capacity By:'||ABS(V_vacancies)
                                  ||'employees.');
  END IF;
  EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred:'
                                  ||SQLERRM);
END;
```

PROGRAM 10

Write a PL/SQL program to count number of employees in a specific department and check whether this department have any vacancies or not. If any vacancies, how many vacancies are in that department.

```
DECLARE
    V_target_dept_id employees.department_id %TYPE:=80;
    V_dept_total_capacity NUMBER:=25;
    V_employee_count NUMBER;
    V_vacancies NUMBER;
BEGIN
    Select COUNT(employee_id) INTO V_employee_count
    FROM employees where department_id = V_target_dept_id;
    V_vacancies := V_dept_total_capacity - V_employee_count;
    DBMS_OUTPUT.PUT_LINE('Department'||V_target_dept_id
    ||'Vacancy Check');
    DBMS_OUTPUT.PUT_LINE('Total Department Capacity:'||
    V_dept_total_capacity);
    DBMS_OUTPUT.PUT_LINE('Current Employee Count:'||
    V_employee_count);
    IF V_vacancies>0 THEN
    DBMS_OUTPUT.PUT_LINE('Vacancy Check: YES, the
    department HAS vacancies.');
    DBMS_OUTPUT.PUT_LINE('Total Vacancies Remaining:'||V_vacancies)
    ELSE
        DBMS_OUTPUT.PUT_LINE('Vacancy Check: NO, the department
        has NO vacancies');
    IF V_vacancies<0 THEN
        DBMS_OUTPUT.PUT_LINE('It is Over capacity by:'||ABS
        (V_vacancies)||'employees.');
```

PROGRAM 11

Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

```
DECLARE
    CURSOR employee_cursor IS
        select employee_id, first_name || ' ' || last_name
AS full_name, job_title, hire_date, salary FROM
employees ORDER BY employee_id;
BEGIN
    DBMS_OUTPUT.PUT_LINE( RPAD('ID',5) || RPAD
('Name',20) || RPAD('Job Title',15) || RPAD('Hire Date',12) ||
'Salary');
FOR emp_rec IN employee_cursor
  LOOP
        DBMS_OUTPUT.PUT_LINE( RPAD( emp_rec.employee_id,
5) || RPAD(emp_rec.full_name,20) || RPAD(emp_rec.job_title,15) ||
TO_CHAR(emp_rec.hire_date,'YYYY-MM-DD') || ' ' || TO_CHAR
(emp_rec.salary,'$99,999.00'); END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employees
found in the table ~~department data found~~.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred:' ||
SQLERRM);
END;
```

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

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

```
DECLARE
  CURSOR dept_employee_cursor IS
    SELECT
      e.employee_id,
      e.first_name || ' ' || e.last_name AS full_name,
      d.department_name
    FROM
      employees e
    JOIN
      departments d ON
  e.department_id = d.department_id
        ORDER BY
          e.employee_id;
  BEGIN
    DBMS_OUTPUT.PUT_LINE(RPAD('ID',5)||RPAD('Name',25)||
  'Department Name');
  FOR emp_dept_rec IN
   dept_employee_cursor LOOP
        DBMS_OUTPUT.PUT_LINE (RPAD(emp_dept_rec.
employee_id,5)||RPAD(emp_dept_rec.full_name,25)||
emp_dept_rec.department_name);
  ENDLOOP;
EXCEPTION
      when NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE ('No employee or department
  data found.');
```

PROGRAM 13

Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

```
DECLARE
    CURSOR job_cursor IS
        select job_id, job_title, min_salary FROM
jobs ORDER BY job_id;
BEGIN
    DBMS_OUTPUT.PUT_LINE(RPAD('Job ID',10)||RPAD('Job Title',
25)||'Min Salary');
FOR job_rec IN job_cursor LOOP
    DBMS_OUTPUT.PUT_LINE(RPAD(job_rec.job_id,10)||
RPAD(job_rec.job_title,25)||TO_CHAR(job_rec.min_salary,
'$99,999'));
END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No job records found
in the table.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error
occurred:' || SQLERRM);
END;
```

PROGRAM 14

Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

```
DECLARE
    CURSOR job_history_cursor IS
        Select e.employee_id, e.first_name || ' ' || e.last_name
AS full_name, jh.start_date FROM employees e JOIN
job_history jh ON e.employee_id = jh.employee_id ORDER BY
e.employee_id, jh.start_date DESC;
BEGIN
  DBMS_OUTPUT.PUT_LINE(RPAD('ID',5) || RPAD('Name',25) ||
'Job History Start date');
  FOR hist_rec IN job_history_cursor
LOOP
    DBMS_OUTPUT.PUT_LINE(RPAD(hist_rec.employee_id,5) ||
RPAD(hist_rec.full_name,25) || TO_CHAR(hist_rec.start_date,
'YYYY-MM-DD'));
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No employee job history
records found.');
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error
occurred:' || SQLERRM);
  END;
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

```
DECLARE
    CURSOR end_date_cursor IS
        Select e.employee_id, e.first_name || ' ' || e.last_name
AS full_name, jh.end_date FROM employees e JOIN
job_history jh ON e.employee_id = jh.employee_id
ORDER BY e.employee_id, jh.end_date DESC;
BEGIN
    DBMS_OUTPUT.PUT_LINE(RPAD('ID',5)|| RPAD('Name',25)||
    'Job History End date');
FOR hist_rec IN end_date_
cursor LOOP
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	5
Program/Execution (5)	5
Viva(5)	3
Total (15)	13
Faculty Signature	

```
    DBMS_OUTPUT.PUT_LINE(RPAD(hist_rec.employee_id,5)||
RPAD(hist_rec.full_name,25)|| TO_CHAR(hist_rec.end_date
'YYYY-MM-DD')); ENDLOOP;
EXCEPTION WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No employee job
history records found!');
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred:'||
SQLERRM);
END;
```

Date : 8/10/25

EXERCISE-16

PROCEDURES AND FUNCTIONS

AIM: To learn about Procedures and Functions in PL/SQL

PROCEDURES

DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part
- Optional exception handling part

These procedures and functions do not show the errors.

KEYWORDS AND THEIR PURPOSES

REPLACE: It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.

IN: Specifies that a value for the argument must be specified when calling the procedure ie. used to pass values to a sub-program. This is the default parameter.

OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

INOUT: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

RETURN: It is the datatype of the function's return value because every function must return a value, this clause is required.

PROCEDURES – SYNTAX

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

FUNCTIONS – SYNTAX

```
create or replace function <function name> (argument in datatype,……) return datatype {is,as}
variable declaration;
```

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
CREATE OR REPLACE FUNCTION CALCULATE_FACTORIAL
(P_number IN NUMBER) RETURN NUMBER IS
V_factorial NUMBER:=1;
BEGIN
  IF P_number < 0 OR P_number != TRUNC(P_number) THEN
  RETURN NULL;
ELSE IF P_number = 0 THEN RETURN 1;
ELSE
      FOR i IN 1..P_number LOOP
          V_factorial := V_factorial *i;
      END LOOP;
      RETURN V_factorial;
    END IF;
EXCEPTION
   WHEN OTHERS THEN
        RETURN NULL;
   END CALCULATE_FACTORIAL;
DECLARE
    V_input_number NUMBER := 5;
    V_result. NUMBER;
BEGIN
    V_result := CALCULATE_FACTORIAL (V_input_number);
IF V_result is NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE ('The factorial of' ||
V_input_number || 'is' || V_result);
ELSE
    DBMS_OUTPUT.PUT_LINE ('cannot calculate factorial
    for the input:' || V_input_number);
  END IF;
END;
```

Program 2

Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library

```
CREATE OR REPLACE PROCEDURE RETRIEVE_
BOOK_INFO( p_book_id IN books.book_id %TYPE,
p_book_title OUT books.title %TYPE, p_copies_check IN OUT
books.copies_available %TYPE --IN: copies to check out;
OUT: New available count) IS v_initial_copies_available
p_book_title := 'INSUFFICIENT COPIES';
RAISE_APPLICATION_ERROR(-20001, 'Insufficient
copies available for book' || p_book_id);
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
p_book_title := 'BOOK NOT FOUND';
p_copies_check := -1
RAISE_APPLICATION_ERROR(-20002, 'Book ID' || p_book_id ||
'not found.'); WHEN OTHERS THEN
p_book_title := 'ERROR';
p_copies_check := -2;
ROLLBACK;
RAISE;
END RETRIEVE_BOOK_
INFO;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	5
Program/Execution (5)	5
Viva(5)	3
Total (15)	13
Faculty Signature	

RESULT:

Thus the concept of procedures and Functions in PL/SQL is studied.