22AIE303 PL-SQL Lab– 1

Name : Anuvind M P
Roll No : AM.EN.U4AIE22010

Question 1

Do all the questions discussed in the class

1. Write a PLSQL program which accepts 2 integer values and return their sums

```
create function Sum_(a int, b int)
returns int as
$$
declare
    c int;
begin
    c := a + b;
    return c;
end;
$$
language 'plpgsql';

select Sum_(5, 10) as result;
```

Data Output | Messages | Notifications

SQL | Showing rows:

	result integer
1	15

```
create function Sum_1(int, int)
returns int as
$$
begin
    return $1+ $2;
end;
$$
language 'plpgsql';

select Sum_1(5, 10) as result;
```

Data Output | Messages | Notifications

SQL

	result integer
1	15

2. Consider the relation **Emp(eno,enam,sal)**

   Create a function which accepts an employee no and returns the sal of that employee

```sql
create table Emp (
    eno int primary key,
    ename varchar(50),
    sal int
);

insert into Emp (eno, ename, sal) values
(101, 'Alice', 75000),
(102, 'Bob', 60000),
(103, 'Charlie', 55000),
(104, 'Diana', 50000),
(105, 'Eve', 45000);
```

```sql
create function emp_sal(emp_no Emp.eno%type)
returns int as
$$
declare
    sal1 Emp.sal%type;
begin
    select sal into sal1 from Emp where eno = emp_no;
    return sal1;
end;
$$
language 'plpgsql';

select emp_sal(101) as employee_salary;
```

Data Output | Messages | Notifications

Showing rows:

	employee_salary integer
1	75000

3. Consider the relation **Emp(eno,ename,sal,dno)**

   Create a function which accepts eno and returns the salary and dept number of that employee

   (Practice IN OUT INOUT)

```sql
create table Emp (
    eno int primary key,
    ename varchar(50),
    job varchar(50),
    sal int
);

insert into Emp (eno, ename, job, sal) values
(101, 'Alice', 'Manager', 75000),
(102, 'Bob', 'Developer', 60000),
(103, 'Charlie', 'Analyst', 55000),
(104, 'Diana', 'Tester', 50000),
(105, 'Eve', 'Support', 45000);
```

```
67  create or replace function emp_details(
68      emp_no Emp.eno%type,
69      out emp_name Emp.ename%type,
70      out emp_sal Emp.sal%type
71  ) as
72  $$
73  begin
74      select ename, sal into emp_name, emp_sal from Emp where eno = emp_no;
75  end;
76  $$
77  language 'plpgsql';
78
79  select * from emp_details(102);
80
```

Data Output | Messages | Notifications

Showing rows:

	emp_name character varying	emp_sal integer
1	Bob	60000

4. Consider the relation :

   Salesperson(sl_no, sl_name, sal, commn)

   Product(pno, pname, unit_price)

   Sales(sno, sl_no, pno, qty_sold)

   Write a function which accepts a sales person number and check if the salesperson is eligible for getting commission or not

   A person is eligible for commission only if he has sold a total of 1000 quantity or more for all the products together. If the salesperson is eligible, update the commission amount for the corresponding salesperson in the salesperson table the Commission column in the salesperson table is initially null

```sql
create table Sales (
    sno int primary key,
    slno int references Salesperson(slno),
    pno int references Product(pno),
    qty_sold int
);

insert into Salesperson (slno, slname, sal, commn) values
(1, 'Alice', 50000, null),
(2, 'Bob', 45000, null),
(3, 'Charlie', 40000, null);

insert into Product (pno, pname, unit_price) values
(101, 'Laptop', 80000),
(102, 'Mouse', 500),
(103, 'Keyboard', 1000);

insert into Sales (sno, slno, pno, qty_sold) values
(1, 1, 101, 300),
(2, 1, 102, 200),
(3, 1, 103, 600),
(4, 2, 102, 500),
(5, 2, 103, 400),
(6, 3, 101, 700),
(7, 3, 103, 200);
```

```sql
create or replace function eligibility_chk(
    sales_no Salesperson.slno%type,
    out msg varchar
) as
$$
declare
    tot_qty Sales.qty_sold%type;
begin
    select sum(qty_sold) into tot_qty from Sales where slno = sales_no;

    if tot_qty > 1000 then
        update Salesperson
        set commn = 'Y'
        where slno = sales_no;
        msg := 'Eligible';
    else
        update Salesperson
        set commn = 'F'
        where slno = sales_no;
        msg := 'Not Eligible';
    end if;
end;
$$
language 'plpgsql';
```

```
149  select eligibility_chk(1) as status;
150  select eligibility_chk(2) as status;
151  select eligibility_chk(3) as status;
152
153  select * from Salesperson
154
```

Data Output | Messages | Notifications

Showing rows:

	slno [PK] integer	slname character varying (50)	sal integer	commn character (1)
1	1	Alice	50000	Y
2	2	Bob	45000	F
3	3	Charlie	40000	F

5. Assume for all the courses max 60 Students can be registered while a function to register a student for a particular course only if the current number of students registered for that course is not exceeding the limit

```
CREATE TABLE Student (
    sno int PRIMARY KEY,
    sname varchar(50)
);
CREATE TABLE Course (
    cno int PRIMARY KEY,
    cname varchar(50)
);
CREATE TABLE Stud_Course (
    sno int REFERENCES Student(sno),
    cno int REFERENCES Course(cno),
    PRIMARY KEY (sno, cno)
);
INSERT INTO Student (sno, sname) VALUES
(1, 'Alice'),
(2, 'Bob'),
(3, 'Charlie'),
(4, 'Diana'),
(5, 'Eve');
INSERT INTO Course (cno, cname) VALUES
(101, 'Mathematics'),
(102, 'Physics'),
(103, 'Chemistry');
INSERT INTO Stud_Course (sno, cno) VALUES
```

```
(1, 101),
(2, 101),
(3, 101),
(4, 102),
(5, 102);

CREATE OR REPLACE FUNCTION register_student(
    student_no Student.sno%TYPE,
    course_no Course.cno%TYPE
) RETURNS varchar AS
$$
DECLARE
    student_count int;
    already_registered boolean;
BEGIN
    SELECT EXISTS (
        SELECT 1
        FROM Stud_Course
        WHERE sno = student_no AND cno = course_no
    ) INTO already_registered;

    IF already_registered THEN
        RETURN 'Registration Failed: Student already registered for the course';
    END IF;
    SELECT COUNT(*) INTO student_count
    FROM Stud_Course
    WHERE cno = course_no;
    IF student_count < 60 THEN
        INSERT INTO Stud_Course (sno, cno) VALUES (student_no, course_no);
        RETURN 'Registration Successful';
    ELSE
        RETURN 'Registration Failed: Course is full';
    END IF;
END;
$$
LANGUAGE 'plpgsql';
```

SELECT register_student(3, 101) AS registration_status;

Question 2

Consider the following schema .

Emp(eno, ename, sal)

EMP_PROJ(eno, pno, prjt_hrs).

```
CREATE TABLE Emp (
    eno int PRIMARY KEY,
    ename varchar(50),
    sal int
);
CREATE TABLE EMP_PROJ (
    eno int REFERENCES Emp(eno),
    pno int,
    prjt_hrs int,
    PRIMARY KEY(eno,pno)
);
```

```
INSERT INTO Emp (eno, ename, sal) VALUES
(101, 'Alice', 50000),
(102, 'Bob', 45000),
(103, 'Charlie', 40000);

INSERT INTO EMP_PROJ (eno, pno, prjt_hrs) VALUES
(101, 1, 40),
(101, 2, 35),
(102, 1, 20),
(103, 3, 50),
(101, 3, 25),
(102, 2, 30);
```

a) Write a function ProjectLoad that returns the total project working hours for the given eno.

```sql
CREATE OR REPLACE FUNCTION ProjectLoad(
    employee_no Emp.eno%TYPE
) RETURNS int AS
$$
DECLARE
    total_hours int;
BEGIN
    SELECT COALESCE(SUM(prjt_hrs), 0) INTO total_hours
    FROM EMP_PROJ
    WHERE eno = employee_no;

    RETURN total_hours;
END;
$$
LANGUAGE 'plpgsql';

SELECT ProjectLoad(101) AS total_hours;
```

Data Output | Messages | Notifications

	total_hours integer
1	100

b) Write a PL/SQL code to update the salary of an employee if the employee earn less than the average salary.

New salary is current sal + difference between currrent sal and average salary

```
45  UPDATE Emp SET sal=5000 WHERE ename='Bob';
46
47  CREATE OR REPLACE FUNCTION UpdateSalary()
48  RETURNS void AS
49  $$
50  DECLARE
51      avg_sal numeric;
52  BEGIN
53      SELECT AVG(sal) INTO avg_sal FROM Emp;
54
55      UPDATE Emp
56      SET sal = sal + (avg_sal - sal)
57      WHERE sal < avg_sal;
58      RAISE NOTICE 'Salaries updated successfully.';
59  END;
60  $$
61  LANGUAGE 'plpgsql';
62
63
64  SELECT UpdateSalary();
65
66  SELECT* FROM emp
67
```

Data Output | Messages | Notifications

	eno [PK] integer	ename character varying (50)	sal integer
1	101	Alice	50000
2	103	Charlie	45000
3	102	Bob	42778

Question 3

Consider the following tables

Item(ino, iname, unit_price)

Transaction(tr_no,ino,qty)

```sql
CREATE TABLE Item (
    ino int PRIMARY KEY,
    iname varchar(50),
    unit_price int
);
CREATE TABLE Transaction (
    tr_no int,
    ino int REFERENCES Item(ino),
    qty int ,
    PRIMARY KEY(tr_no,ino)
);
```

```sql
INSERT INTO Item (ino, iname, unit_price) VALUES
(101, 'Laptop', 80000),
(102, 'Mouse', 500),
(103, 'Keyboard', 1000),
(104, 'Monitor', 15000),
(105, 'Printer', 12000);

INSERT INTO Transaction (tr_no, ino, qty) VALUES
(1, 101, 2),
(2, 102, 1),
(3, 103, 3),
(4, 104, 5),
(5, 105, 1),
(6, 101, 1),
(7, 104, 2);
```

Write a function to accept an item no from the user. If transaction has been made for less than 2 times for that item (check from transaction table), delete the item from the Item table.

```sql
CREATE OR REPLACE FUNCTION RemoveLowTransactions(
    item_no Item.ino%TYPE
) RETURNS varchar AS
$$
DECLARE
    transaction_count int;
```

```
BEGIN
    SELECT COUNT(*) INTO transaction_count
    FROM Transaction
    WHERE ino = item_no;

    IF transaction_count < 2 THEN
        DELETE FROM Item WHERE ino = item_no;
        RETURN 'Item removed successfully.';
    ELSE
        RETURN 'Item not removed: More than 2 transactions.';
    END IF;
END;
$$
LANGUAGE 'plpgsql';

SELECT RemoveLowTransactions(101) AS result;
```

Data Output | Messages | Notifications

	result character varying
1	Item not removed: More than 2 transactions.

Question 4

Consider a Bank database which includes the following tables.

ACCOUNTS(ac_no,br_no, cust_no,ac_type,bal)

BRANCHES(br_no, br_name, loc)

CUSTOMER(cno, cname, c_type)

```
CREATE TABLE ACCOUNTS (
    ac_no int PRIMARY KEY,
    br_no int,
    cust_no int,
    ac_type varchar(20),
    bal int
);
CREATE TABLE BRANCHES (
    br_no int PRIMARY KEY,
    br_name varchar(50),
    loc varchar(50)
);
CREATE TABLE CUSTOMER (
    cno int PRIMARY KEY,
    cname varchar(50),
    c_type varchar(20)
);
```

```sql
INSERT INTO ACCOUNTS (ac_no, br_no, cust_no, ac_type, bal) VALUES
(1001, 101, 201, 'Savings', 60000),
(1002, 101, 202, 'Current', 40000),
(1003, 102, 201, 'Savings', 30000),
(1004, 102, 203, 'Savings', 80000),
(1005, 103, 204, 'Current', 20000);
INSERT INTO BRANCHES (br_no, br_name, loc) VALUES
(101, 'Main Branch', 'City Center'),
(102, 'East Branch', 'Downtown'),
(103, 'West Branch', 'Uptown');
INSERT INTO CUSTOMER (cno, cname, c_type) VALUES
(201, 'Alice', 'Class A'),
(202, 'Bob', 'Class B'),
(203, 'Charlie', 'Class B'),
(204, 'Diana', 'Class B');
```

a) Write a function that accepts a threshold value and a customer number. The program updates the c_type based on the threshold value. Ie. If balance > threshold then class A, else class B.

```sql
CREATE OR REPLACE FUNCTION UpdateCustomerType(
    threshold_val int,
    customer_no CUSTOMER.cno%TYPE
) RETURNS varchar AS
$$
DECLARE
    total_balance int;
BEGIN
    SELECT SUM(bal) INTO total_balance
    FROM ACCOUNTS
    WHERE cust_no = customer_no;

    IF total_balance > threshold_val THEN
        UPDATE CUSTOMER
        SET c_type = 'Class A'
        WHERE cno = customer_no;
        RETURN 'Customer updated to Class A.';
    ELSE
        UPDATE CUSTOMER
        SET c_type = 'Class B'
        WHERE cno = customer_no;
        RETURN 'Customer updated to Class B.';
    END IF;
END;
$$
LANGUAGE 'plpgsql';
```

```sql
SELECT UpdateCustomerType(50000, 101) AS result;
```

Data Output | Messages | Notifications

	result character varying
1	Customer updated to Class B.

b) Write a function called CloseBranch that takes two arguments (the branch to be closed and the branch to take over the accounts) and transfers all accounts at the closing branch to the new branch and removes the closing branch.

```
67  CREATE OR REPLACE FUNCTION CloseBranch(
68      closing_branch BRANCHES.br_no%TYPE,
69      receiving_branch BRANCHES.br_no%TYPE
70  ) RETURNS varchar AS
71  $$
72  BEGIN
73      UPDATE ACCOUNTS
74      SET br_no = receiving_branch
75      WHERE br_no = closing_branch;
76
77      DELETE FROM BRANCHES WHERE br_no = closing_branch;
78
79      RETURN 'Branch closed and accounts transferred successfully.';
80  END;
81  $$
82  LANGUAGE 'plpgsql';
83
84  SELECT CloseBranch(101, 102) AS result;
85
86
87
```

Data Output | Messages | Notifications

	result character varying
1	Branch closed and accounts transferred successfull...

c) Write a function that implements a "safe" withdrawal operation, that only permits a withdraw if there are sufficient funds in the account to cover it.

```
88  CREATE OR REPLACE FUNCTION SafeWithdraw(
89      account_no ACCOUNTS.ac_no%TYPE,
90      withdraw_amount int
91  ) RETURNS varchar AS
92  $$
93  DECLARE
94      current_balance int;
95  BEGIN
96      SELECT bal INTO current_balance
97      FROM ACCOUNTS
98      WHERE ac_no = account_no;
99
100     IF current_balance >= withdraw_amount THEN
101         UPDATE ACCOUNTS
102         SET bal = bal - withdraw_amount
103         WHERE ac_no = account_no;
104         RETURN 'Withdrawal successful.';
105     ELSE
106         RETURN 'Withdrawal failed: Insufficient balance.';
107     END IF;
108 END;
109 $$
110 LANGUAGE 'plpgsql';
111
112 SELECT SafeWithdraw(1001, 5000) AS result;
113
```

Data Output | Messages | Notifications

SQL

	result character varying
1	Withdrawal successful.