

---

# DATABASE MANGEMENT SYSTEMS

---

## PLSQL Lab1

1. Write a function that accepts two integers as inputs and returns the sum of integers

```
1 create function sumint(a int, b int, out c int) as
2 $$
3 begin
4 c:=a+b;
5 end;
6 $$
7 Language 'plpgsql';
8
9 select sumint(5, 4);
```

```
sumint
-----
          9
(1 row)
```

2. Write a function that accepts the employee no as input and returns the salary of that employee

```

11 CREATE TABLE employee (empno Integer PRIMARY KEY,
12  ename VARCHAR(20) NOT NULL,
13  job varchar(10),
14  mgr_id Integer,
15  hired_date DATE,
16  Basic_sal Numeric(6,2) DEFAULT 1000);
17
18 INSERT INTO employee VALUES (7369, 'Smith', 'Clerk', 7902, '1980-12-17', 6800),
19 (7499, 'Allen', 'Salesman', 7698, '1981-02-20', 1160),
20 (7521, 'Ward', 'Salesman', 7698, '1981-02-22', 1125),
21 (7566, 'Jones', 'Manager', 7839, '1981-04-02', 2297),
22 (7654, 'Martin', 'Salesman', 7698, '1981-09-28', 1125),
23 (7698, 'Blake', 'Manager', 7839, '1981-05-01', 2285),
24 (7782, 'Clark', 'Manager', 7839, '1981-06-09', 2245),
25 (7788, 'Scott', 'Analyst', 7566, '1982-12-09', 1300);
26
27 CREATE function Empsal(emp_no employee.empno%type, out sal employee.Basic_sal%type) as'
28 begin
29 select Basic_sal INTO sal from employee where empno=emp_no;
30 end;
31 'Language "plpgsql";
32
33 select Empsal(7521);

```

```

empсал
-----
1125.00
(1 row)

```

3. Write a function that accepts the employee no of an employee and returns the salary and department no as output

```

36 CREATE TABLE Emp (emp_no Integer PRIMARY KEY,
37  ename VARCHAR(20) NOT NULL,
38  Basic_sal Integer DEFAULT 1000,
39  dept_no Integer);
40
41 INSERT INTO Emp VALUES (7369, 'Smith', 45000, 1),
42 (7499, 'Allen', 20000, 1),
43 (7521, 'Ward', 19000, 2),
44 (7566, 'Jones', 48000, 1),
45 (7654, 'Martin', 15000, 3),
46 (7698, 'Blake', 32000, 3),
47 (7782, 'Clark', 20000, 2),
48 (7788, 'Scott', 10000, 4);

```

```
50 CREATE function EmpDetails(inout empno EMP.emp_no%type,  
51 out sal EMP.Basic_sal%type,  
52 out deptno EMP.dept_no%type) as '  
53 begin  
54 select Basic_sal, dept_no into sal, deptno from EMP where emp_no=empno;  
55 end;  
56 'Language "plpgsql";  
57  
58 select EmpDetails(7782);  
  
empdetails  
-----  
      (7782,20000,2)  
(1 row)
```

4. Write a function that accepts sales person id as input and check if that sales person is eligible for commission, eligibility criteria for commission is that the salesperson must have sold more than 1000 products combined all products together. If eligible, update the commission entry to Y.

```
61 CREATE TABLE SalesPerson (  
62     slno INT PRIMARY KEY,  
63     slname VARCHAR(10),  
64     commission CHAR(1)  
65 );  
66 CREATE TABLE Product (  
67     pno INT PRIMARY KEY,  
68     pname VARCHAR(10),  
69     unitprice INT  
70 );  
71 CREATE TABLE Sales (  
72     sno INT PRIMARY KEY,  
73     slno INT REFERENCES SalesPerson(slno),  
74     pno INT REFERENCES Product(pno),  
75     qtysold int  
76 );  
77 INSERT INTO SalesPerson (slno, slname, commission) VALUES  
78 (1, 'Alice', null),  
79 (2, 'Bob', null),  
80 (3, 'Charlie', null);  
81 INSERT INTO Product (pno, pname, unitprice) VALUES  
82 (101, 'Laptop', 800),  
83 (102, 'Tablet', 500),  
84 (103, 'Phone', 300);  
85 INSERT INTO Sales (sno, slno, pno, qtysold) VALUES  
86 (1, 1, 101, 6501),  
87 (2, 2, 103, 540),  
88 (3, 3, 102, 1200);  
89  
90 CREATE function commission_eligible(sales_no SalesPerson.slno%type, OUT msg varchar) AS  
91 $$  
92 Declare  
93 tot_qty Sales.qtysold%type;  
94  
95 begin  
96 select sum(qtysold) into tot_qty from Sales where slno = sales_no;  
97 if tot_qty > 1000 then  
98 update SalesPerson set commission = 'y' where slno = sales_no;  
99 msg := 'Eligible';  
100  
101 else  
102 update SalesPerson set commission = 'n' where slno = sales_no;  
103 msg := 'Not Eligible';  
104  
105 end if;  
106  
107 end;  
108 $$  
109  
110 language 'plpgsql';  
111  
112 select commission_eligible(1);  
113 select commission_eligible(2);
```

```

commission_eligible
-----
Eligible
(1 row)

```

```

commission_eligible
-----
Not Eligible
(1 row)

```

5. For all courses maximum sixty students can be registered write a function to register a student for a particular course only if current Number of students registered for that course is not exceeding the limit.

```

1 CREATE TABLE Student (
2     sno INT PRIMARY KEY,
3     sname VARCHAR(15)
4 );
5 CREATE TABLE Course (
6     cno INT PRIMARY KEY,
7     cname VARCHAR(15)
8 );
9 CREATE TABLE Stud_Course (
10    sno INT REFERENCES Student(sno),
11    cno INT REFERENCES Course(cno),
12    PRIMARY KEY (sno, cno)
13 );
14 INSERT INTO Student (sno, sname) VALUES
15 (1, 'Alice'),
16 (2, 'Bob'),
17 (3, 'Charlie'),
18 (4, 'Diana'),
19 (5, 'Eve');
20 INSERT INTO Course (cno, cname) VALUES
21 (101, 'Mathematics'),
22 (102, 'Physics'),
23 (103, 'Chemistry');

```

```

25 CREATE OR REPLACE FUNCTION register_student(s_id INT, c_id INT)
26 RETURNS TEXT AS $$
27 DECLARE
28     current_count INT;
29 BEGIN
30     SELECT COUNT(*) INTO current_count
31     FROM Stud_Course
32     WHERE cno = c_id;
33     IF current_count >= 60 THEN
34         RETURN 'Registration failed: Maximum student limit (60) reached for this course.';
35     ELSE
36         INSERT INTO Stud_Course (sno, cno) VALUES (s_id, c_id);
37         RETURN 'Registration successful!';
38     END IF;
39 END;
40 $$ LANGUAGE plpgsql;
41 SELECT register_student(1, 101);
42 SELECT register_student(2, 101);
43 SELECT * FROM Stud_Course;

```

```

      register_student
-----
Registration successful!
(1 row)

```

```

      register_student
-----
Registration successful!
(1 row)

```

```

      sno | cno
-----+-----
       1 | 101
       2 | 101
(2 rows)

```

## 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.

```
26 CREATE OR REPLACE FUNCTION ProjectLoad(  
27     employee_no Emp.eno%TYPE  
28 ) RETURNS int AS  
29 $$  
30 DECLARE  
31     total_hours int;  
32 BEGIN  
33     SELECT COALESCE(SUM(prjt_hrs), 0) INTO total_hours  
34     FROM EMP_PROJ  
35     WHERE eno = employee_no;  
36  
37     RETURN total_hours;  
38 END;  
39 $$  
40 LANGUAGE 'plpgsql';  
41  
42 SELECT ProjectLoad(101) AS total_hours;  
43
```

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  
CURRENT 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)

```
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)  
);
```

```
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.

```
30 CREATE OR REPLACE FUNCTION RemoveLowTransactions(  
31     item_no Item.ino%TYPE  
32 ) RETURNS varchar AS  
33 $$  
34 DECLARE  
35     transaction_count int;
```

```
36 BEGIN
37     SELECT COUNT(*) INTO transaction_count
38     FROM Transaction
39     WHERE ino = item_no;
40
41     IF transaction_count < 2 THEN
42         DELETE FROM Item WHERE ino = item_no;
43         RETURN 'Item removed successfully.';
44     ELSE
45         RETURN 'Item not removed: More than 2 transactions.';
46     END IF;
47 END;
48 $$
49 LANGUAGE 'plpgsql';
50
51 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)
);
```

```
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.

```
36 CREATE OR REPLACE FUNCTION UpdateCustomerType(
37     threshold_val int,
38     customer_no CUSTOMER.cno%TYPE
39 ) RETURNS varchar AS
40 $$
41 DECLARE
42     total_balance int;
43 BEGIN
44     SELECT SUM(bal) INTO total_balance
45     FROM ACCOUNTS
46     WHERE cust_no = customer_no;
47
48     IF total_balance > threshold_val THEN
49         UPDATE CUSTOMER
50         SET c_type = 'Class A'
51         WHERE cno = customer_no;
52         RETURN 'Customer updated to Class A.';
53     ELSE
54         UPDATE CUSTOMER
55         SET c_type = 'Class B'
56         WHERE cno = customer_no;
57         RETURN 'Customer updated to Class B.';
58     END IF;
59 END;
60 $$
61 LANGUAGE 'plpgsql';
```

```

63 SELECT UpdateCustomerType(50000, 101) AS result;
64
65

```

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  
114
```

Data Output Messages Notifications



	result	lock icon
	character varying	
1	Withdrawal successful.	