SUBJECT: CA 504 MJP Lab Course Based on CA 501 MJ & CA 503 MJ (Database Systems and SQL, Operating Systems)

Assignment 1

Q.1) Consider the following database:

Room (room_no, room_name, room_type, charges)

Guest (Guest_code, Gname, city, no_of_persons)

The relationship is **one-to-one** between Room and Guest.
The room_type can be 'AC' or 'NonAC'.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE HotelDB;

\c HotelDB;

-- Create Room Table
CREATE TABLE Room (
    room_no SERIAL PRIMARY KEY,
    room_name VARCHAR(50),
    room_type VARCHAR(10) CHECK (room_type IN ('AC', 'NonAC')),
    charges NUMERIC(10,2)
);

-- Create Guest Table
CREATE TABLE Guest (
    Guest_code SERIAL PRIMARY KEY,
    Gname VARCHAR(50),
    city VARCHAR(50),
    no_of_persons INT,
```

```
    room_no INT UNIQUE REFERENCES Room(room_no)
);

-- Insert Records into Room
INSERT INTO Room (room_name, room_type, charges)
VALUES
('Rose Suite', 'AC', 2500),
('Jasmine Room', 'NonAC', 1500),
('Lotus Room', 'AC', 3000),
('Tulip Room', 'NonAC', 1200);

-- Insert Records into Guest
INSERT INTO Guest (Gname, city, no_of_persons, room_no)
VALUES
('Sanjay Mehta', 'Pune', 2, 1),
('Anjali Sharma', 'Mumbai', 3, 2),
('Sameer Khan', 'Delhi', 1, 3),
('Priya Patel', 'Ahmedabad', 4, 4);
```

---

i) List all guests whose name starts with “S”.

```
SELECT * FROM Guest
WHERE Gname LIKE 'S%';
```

OUTPUT:-

```
 guest_code |    gname     |  city  | no_of_persons | room_no
------------+--------------+--------+---------------+---------
    1       | Sanjay Mehta | Pune   |       2       |   1
    3       | Sameer Khan  | Delhi  |       1       |   3
```

---

ii) Increase the charges of all AC rooms by 15%.

```
UPDATE Room
SET charges = charges * 1.15
```

WHERE room_type = 'AC';

OUTPUT:-

UPDATE 2

(Charges of AC rooms increased by 15%)

---

iii) List the minimum charges of a room.

SELECT MIN(charges) AS Minimum_Charges FROM Room;

OUTPUT:-

minimum_charges

1200.00

---

iv) List the names of the guests in the sorted order by city name.

SELECT Gname, city FROM Guest

ORDER BY city ASC;

OUTPUT:-

gname | city

---+---

Priya Patel | Ahmedabad

Sameer Khan | Delhi

Anjali Sharma | Mumbai

Sanjay Mehta | Pune

B) Write a procedure to find sum and product of two numbers. [10 Marks]

Solution:

```
CREATE OR REPLACE PROCEDURE sum_and_product(
    a INT,
    b INT,
    OUT sum_result INT,
    OUT product_result INT
)
LANGUAGE plpgsql
AS $$
```

```
BEGIN
    sum_result := a + b;
    product_result := a * b;
END;
$$;
```

Call the Procedure (I/P):

```
CALL sum_and_product(7, 3, sum_result, product_result);
```

Output (O/P):

```
 sum_result | product_result
------------+---------------
     10     |       21
```

Q.2) Write a program to implement FCFS CPU scheduling algorithm. Take arrival time, burst time for n number of processes from the user. Calculate average waiting time. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, j;
   float avg_wait = 0, avg_turn = 0;
   int bt[20], at[20], wt[20], tat[20], ct[20];


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for(i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
      scanf("%d%d", &at[i], &bt[i]);
   }


   ct[0] = at[0] + bt[0];
   tat[0] = ct[0] - at[0];
   wt[0] = tat[0] - bt[0];
```

```
    for(i = 1; i < n; i++) {
        if(ct[i-1] < at[i])
            ct[i] = at[i] + bt[i];
        else
            ct[i] = ct[i-1] + bt[i];

        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
    }

    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for(i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
        avg_wait += wt[i];
        avg_turn += tat[i];
    }

    printf("\nAverage Waiting Time = %.2f", avg_wait/n);
    printf("\nAverage Turnaround Time = %.2f\n", avg_turn/n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 3
Enter Arrival Time and Burst Time for Process 3: 2 8

Process AT      BT      CT      TAT     WT
P1      0       5       5       5       0
```

P2	1	3	8	7	4
P3	2	8	16	14	6

Average Waiting Time = 3.33

Average Turnaround Time = 8.67

Assignment 2

Q.1) Consider the following database:

College (cno, cname, street_name, ccity)

Principal (pno, pname, experience, salary)

The relationship is **College–Principal: one-to-one**.
The **experience** must be **greater than 10 years**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE CollegeDB;


\c CollegeDB;


-- Create Table College
CREATE TABLE College (
    cno SERIAL PRIMARY KEY,
    cname VARCHAR(50),
    street_name VARCHAR(50),
    ccity VARCHAR(50)
);


-- Create Table Principal
CREATE TABLE Principal (
    pno SERIAL PRIMARY KEY,
    pname VARCHAR(50),
    experience INT CHECK (experience > 10),
    salary NUMERIC(10,2),
    cno INT UNIQUE REFERENCES College(cno)
);
```

```
-- Insert Records into College
INSERT INTO College (cname, street_name, ccity)
VALUES
('Sandalwood College', 'MG Road', 'Pune'),
('Greenland College', 'FC Road', 'Mumbai'),
('Crescent and Moon College', 'MG Road', 'Delhi'),
('Diamond College', 'LBS Road', 'Nagpur');

-- Insert Records into Principal
INSERT INTO Principal (pname, experience, salary, cno)
VALUES
('Dr. Suresh Patil', 15, 85000, 1),
('Dr. Anjali Mehta', 18, 90000, 2),
('Dr. Sanjay Kumar', 12, 78000, 3),
('Dr. Neha Sharma', 25, 100000, 4);
```

---

i) Display all colleges whose name contains ‘and’.

```
SELECT * FROM College
WHERE cname ILIKE '%and%';
```

OUTPUT:-

```
 cno |         cname              | street_name |  ccity
-----+----------------------------+-------------+---------
  1  | Sandalwood College         | MG Road     | Pune
  2  | Greenland College          | FC Road     | Mumbai
  3  | Crescent and Moon College  | MG Road     | Delhi
```

---

ii) List the average salary of a Principal.

```
SELECT AVG(salary) AS Average_Salary FROM Principal;
```

OUTPUT:-

```
 average_salary
----------------
```

88375.00

---

iii) List the names of all Principals having experience between 10 to 20 years.

SELECT pname, experience FROM Principal

WHERE experience BETWEEN 10 AND 20;

OUTPUT:-

pname | experience

Dr. Suresh Patil | 15

Dr. Anjali Mehta | 18

Dr. Sanjay Kumar | 12

---

iv) Change the street name of college 'Sandalwood College' from MG Road to Nehru Road.

UPDATE College

SET street_name = 'Nehru Road'

WHERE cname = 'Sandalwood College';

OUTPUT:-

UPDATE 1

(Street name successfully updated)

B) Write a stored procedure to insert a record in table College. [10 Marks]

Solution:

```
-- Create a stored procedure to insert a record into College table
CREATE OR REPLACE PROCEDURE insert_college(
    p_college_id INT,
    p_college_name VARCHAR,
    p_location VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO College(college_id, college_name, location)
    VALUES (p_college_id, p_college_name, p_location);
END;
$$;
```

Call the Procedure (I/P):

```
CALL insert_college(1, 'ABC College', 'Pune');
```

Output (O/P):

```
Query OK, 1 row affected
```

I/P: `1, 'ABC College', 'Pune'`
O/P: Record inserted successfully into `College` table

Q.2) Write a program to implement FCFS CPU scheduling algorithm. Take arrival time and burst time for n number of processes from the user. Calculate average turnaround time. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i;
   float avg_tat = 0;
   int at[20], bt[20], ct[20], tat[20];


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
      scanf("%d%d", &at[i], &bt[i]);
   }


   ct[0] = at[0] + bt[0];
   tat[0] = ct[0] - at[0];


   for (i = 1; i < n; i++) {
      if (ct[i-1] < at[i])
         ct[i] = at[i] + bt[i];
```

```
        else
            ct[i] = ct[i-1] + bt[i];

        tat[i] = ct[i] - at[i];
    }

    printf("\nProcess\tAT\tBT\tCT\tTAT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i]);
        avg_tat += tat[i];
    }

    printf("\nAverage Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Arrival Time and Burst Time for Process 1: 0 5

Enter Arrival Time and Burst Time for Process 2: 1 3

Enter Arrival Time and Burst Time for Process 3: 2 8

Process	AT	BT	CT	TAT
P1	0	5	5	5
P2	1	3	8	7
P3	2	8	16	14

Average Turnaround Time = 8.67

Assignment 3

Q.1) Consider the following database:

Employee(eno, ename, designation, salary)

Department(dno, dname, location)

The relationship is **Employee–Department: many-to-one.**
The location field **should not be NULL.**

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE CompanyDB;

\c CompanyDB;

-- Create Department Table
CREATE TABLE Department (
    dno SERIAL PRIMARY KEY,
    dname VARCHAR(50),
    location VARCHAR(50) NOT NULL
);

-- Create Employee Table
CREATE TABLE Employee (
    eno SERIAL PRIMARY KEY,
    ename VARCHAR(50),
    designation VARCHAR(30),
    salary NUMERIC(10,2),
    dno INT REFERENCES Department(dno)
);

-- Insert Records into Department
```

```
INSERT INTO Department (dname, location)
VALUES
('HR', 'Pune'),
('Finance', 'Mumbai'),
('IT', 'Delhi'),
('Sales', 'Pune');

-- Insert Records into Employee
INSERT INTO Employee (ename, designation, salary, dno)
VALUES
('Rohit Mehar', 'Manager', 85000, 1),
('Sneha Nair', 'Clerk', 40000, 2),
('Sameer', 'Developer', 78000, 3),
('Ankur', 'Tester', 62000, 3),
('Pranav', 'Sales Executive', 55000, 4);
```

---

i) Give a 5% raise in salary to all the employees.

```
UPDATE Employee
SET salary = salary * 1.05;
```

OUTPUT:-

```
UPDATE 5
(All employees' salaries increased by 5%)
```

---

ii) Display average salary of an employee.

```
SELECT AVG(salary) AS Average_Salary FROM Employee;
```

OUTPUT:-

```
 average_salary
----------------
    65175.00
```

---

iii) List the details of all the departments located at city 'Pune'.

SELECT * FROM Department

WHERE location = 'Pune';

OUTPUT:-

dno | dname | location

-----+--------+----------

1 | HR | Pune

4 | Sales | Pune

---

iv) Display the details of employees whose names end with alphabet “r”.

SELECT * FROM Employee

WHERE ename ILIKE '%r';

OUTPUT:-

eno | ename | designation | salary | dno

-----+----------+-------------+---------+-----

3 | Sameer | Developer | 81900.00| 3

4 | Ankur | Tester | 65100.00| 3

B) Write a stored function using cursors to display all the details of Employee whose salary is more than 80,000. [10 Marks]

Solution:

```
-- Create a stored function using cursor
CREATE OR REPLACE FUNCTION get_high_salary_employees()
RETURNS TABLE(emp_id INT, emp_name VARCHAR, salary NUMERIC, dept VARCHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary, dept
        FROM Employee
        WHERE salary > 80000;
    emp_record RECORD;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN NOT FOUND;
        emp_id := emp_record.emp_id;
        emp_name := emp_record.emp_name;
        salary := emp_record.salary;
        dept := emp_record.dept;
        RETURN NEXT;
```

```
    END LOOP;
    CLOSE emp_cursor;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_high_salary_employees();
```

Output (O/P) Example:

```
 emp_id | emp_name   | salary   | dept
--------+------------+----------+------
 101    | Shivraj       | 90000    | IT
 105    | Alice      | 85000    | HR
```

I/P: Salary threshold is `80,000`
O/P: All employee details with salary > 80,000

Q.2) Write a program to simulate Pre-emptive Shortest Job First (SJF) CPU scheduling algorithm. Accept number of processes, arrival time, and burst time from the user. Calculate and display the average waiting time. [10 Marks]

C Program:

```
#include <stdio.h>

#include <limits.h>


int main() {

   int n, i, completed = 0, time = 0, smallest;

   float avg_wait = 0;

   int at[20], bt[20], rt[20], wt[20], ct[20], tat[20], min;


   printf("Enter number of processes: ");

   scanf("%d", &n);


   for(i = 0; i < n; i++) {

      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);

      scanf("%d%d", &at[i], &bt[i]);

      rt[i] = bt[i];
```

```
}

while (completed != n) {
   smallest = -1;
   min = INT_MAX;

   for (i = 0; i < n; i++) {
      if (at[i] <= time && rt[i] > 0 && rt[i] < min) {
         min = rt[i];
         smallest = i;
      }
   }

   if (smallest == -1) {
      time++;
      continue;
   }

   rt[smallest]--;
   time++;

   if (rt[smallest] == 0) {
      completed++;
      ct[smallest] = time;
      tat[smallest] = ct[smallest] - at[smallest];
      wt[smallest] = tat[smallest] - bt[smallest];
      avg_wait += wt[smallest];
   }
}

printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
```

```
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
            i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
    }

    printf("\nAverage Waiting Time = %.2f\n", avg_wait/n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 7
Enter Arrival Time and Burst Time for Process 2: 2 4
Enter Arrival Time and Burst Time for Process 3: 4 1

Process AT      BT      CT      TAT     WT
P1      0       7       12      12      5
P2      2       4       8       6       2
P3      4       1       5       1       0

Average Waiting Time = 2.33
```

Assignment 4

Q.1) Consider the following database:

Person (pnumber, pname, birthdate, income)

Area (area_code, aname, area_type, pincode)

The relationship is **Person–Area: many-to-one.**
The area_type can have values as either 'urban' or 'rural'.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE CityDB;


\c CityDB;


-- Create Area Table
CREATE TABLE Area (
    area_code SERIAL PRIMARY KEY,
    aname VARCHAR(50),
    area_type VARCHAR(10) CHECK (area_type IN ('urban', 'rural')),
    pincode INT
);


-- Create Person Table
CREATE TABLE Person (
    pnumber SERIAL PRIMARY KEY,
    pname VARCHAR(50),
    birthdate DATE,
    income NUMERIC(10,2),
    area_code INT REFERENCES Area(area_code)
);
```

```
-- Insert Records into Area
INSERT INTO Area (aname, area_type, pincode)
VALUES
('Kalyaninagar', 'urban', 411006),
('Hadapsar', 'urban', 411028),
('Narhe', 'rural', 412115),
('Baner', 'urban', 411045);

-- Insert Records into Person
INSERT INTO Person (pname, birthdate, income, area_code)
VALUES
('Rohan Deshmukh', '1998-03-22', 65000, 1),
('Priya Kulkarni', '2000-06-15', 45000, 2),
('Rahul Patil', '1995-10-05', 72000, 3),
('Neha Joshi', '1999-12-25', 58000, 4),
('Ramesh Pawar', '1990-01-20', 90000, 1);
```

---

i) List the details of all people whose name starts with the alphabet “R”.

```
SELECT * FROM Person
WHERE pname ILIKE 'R%';
```

OUTPUT:-

```
 pnumber |     pname      |  birthdate  | income  | area_code
---------+----------------+-------------+----------+-----------
   1     | Rohan Deshmukh  | 1998-03-22  | 65000.00 | 1
   3     | Rahul Patil     | 1995-10-05  | 72000.00 | 3
   5     | Ramesh Pawar    | 1990-01-20  | 90000.00 | 1
```

---

ii) Display the details of people in the sorted order of their income.

```
SELECT * FROM Person
ORDER BY income ASC;
```

OUTPUT:-

pnumber	pname	birthdate	income	area_code
2	Priya Kulkarni	2000-06-15	45000.00	2
4	Neha Joshi	1999-12-25	58000.00	4
1	Rohan Deshmukh	1998-03-22	65000.00	1
3	Rahul Patil	1995-10-05	72000.00	3
5	Ramesh Pawar	1990-01-20	90000.00	1

---

iii) Display the count of areas of "urban" type.

```
SELECT COUNT(*) AS Urban_Area_Count
FROM Area
WHERE area_type = 'urban';
```

OUTPUT:-

urban_area_count
3

---

iv) Change the pincode of "Kalyaninagar" to 411036.

```
UPDATE Area
SET pincode = 411036
WHERE aname ILIKE 'Kalyaninagar';
```

OUTPUT:-

UPDATE 1

(Pincode updated successfully)

B) Create a stored procedure named as "addrecords" for adding person records. [10 Marks]

Solution:

```
-- Create a stored procedure to add person records
CREATE OR REPLACE PROCEDURE addrecords(
    p_person_id INT,
    p_name VARCHAR,
```

```
    p_age INT,
    p_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Person(person_id, name, age, city)
    VALUES (p_person_id, p_name, p_age, p_city);
END;
$$;
```

Call the Procedure (I/P):

```
CALL addrecords(1, 'Shivraj', 25, 'Pune');
```

Output (O/P):

```
Query OK, 1 row affected
```

I/P: `1, 'Shivraj', 25, 'Pune'`
O/P: Record inserted successfully into `Person` table

Q.2) Write a program to simulate Pre-emptive Shortest Job First (SJF) CPU scheduling algorithm. Accept no. of processes, arrival time, and burst time from the user. Calculate and display the average turnaround time. [10 Marks]

C Program:

```
#include <stdio.h>

#include <limits.h>


int main() {

   int n, i, completed = 0, time = 0, smallest;

   float avg_tat = 0;

   int at[20], bt[20], rt[20], ct[20], tat[20], min;


   printf("Enter number of processes: ");

   scanf("%d", &n);


   for (i = 0; i < n; i++) {

      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
```

```
    scanf("%d%d", &at[i], &bt[i]);
    rt[i] = bt[i];
}

while (completed != n) {
    smallest = -1;
    min = INT_MAX;

    for (i = 0; i < n; i++) {
        if (at[i] <= time && rt[i] > 0 && rt[i] < min) {
            min = rt[i];
            smallest = i;
        }
    }

    if (smallest == -1) {
        time++;
        continue;
    }

    rt[smallest]--;
    time++;

    if (rt[smallest] == 0) {
        completed++;
        ct[smallest] = time;
        tat[smallest] = ct[smallest] - at[smallest];
        avg_tat += tat[smallest];
    }
}
```

```
    printf("\nProcess\tAT\tBT\tCT\tTAT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i]);
    }

    printf("\nAverage Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Arrival Time and Burst Time for Process 1: 0 7

Enter Arrival Time and Burst Time for Process 2: 2 4

Enter Arrival Time and Burst Time for Process 3: 4 1

Process	AT	BT	CT	TAT
P1	0	7	12	12
P2	2	4	8	6
P3	4	1	5	1

Average Turnaround Time = 6.33

Assignment 5

Q.1) Consider the following database:

Doctor (dno, dname, addr, phone_no, specialization)

Patient (pno, pat_name, city, disease)

The relationship is **Doctor–Patient: many-to-many**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE HospitalDB;

\c HospitalDB;

-- Create Doctor Table
CREATE TABLE Doctor (
    dno SERIAL PRIMARY KEY,
    dname VARCHAR(50),
    addr VARCHAR(100),
    phone_no VARCHAR(15),
    specialization VARCHAR(50)
);

-- Create Patient Table
CREATE TABLE Patient (
    pno SERIAL PRIMARY KEY,
    pat_name VARCHAR(50),
    city VARCHAR(50),
    disease VARCHAR(50)
);

-- Create Relationship Table (Many-to-Many)
```

```
CREATE TABLE Doctor_Patient (
   dno INT REFERENCES Doctor(dno),
   pno INT REFERENCES Patient(pno),
   PRIMARY KEY (dno, pno)
);

-- Insert Records into Doctor
INSERT INTO Doctor (dname, addr, phone_no, specialization)
VALUES
('Manoj Deshmukh', 'Alandi', '9876543210', 'Neurologist'),
('Minal Patil', 'Pune', '9823456789', 'Cardiologist'),
('Ramesh Kulkarni', 'Alandi', '9754213698', 'Orthopedic'),
('Meena Joshi', 'Mumbai', 'Neurologist'),
('Suresh Pawar', 'Nashik', '9988776655', 'ENT');

-- Insert Records into Patient
INSERT INTO Patient (pat_name, city, disease)
VALUES
('Ravi Shinde', 'Pune', 'Fever'),
('Sneha Kulkarni', 'Nashik', 'Cold'),
('Rutuja Patil', 'Mumbai', 'Fever'),
('Anil Pawar', 'Nashik', 'Asthma'),
('Manasi Desai', 'Alandi', 'Fever');

-- Insert Records into Doctor_Patient (Relationship)
INSERT INTO Doctor_Patient (dno, pno)
VALUES
(1,1), (1,3), (2,2), (3,4), (4,5), (5,1);
```

---

i) Find the names of all doctors which start with “M”.

```
SELECT dname FROM Doctor
```

```
WHERE dname ILIKE 'M%';
```

OUTPUT:-

```
    dname
------------------
 Manoj Deshmukh
 Minal Patil
 Meena Joshi
```

---

ii) Count the number of doctors who are Neurologists.

```
SELECT COUNT(*) AS Neurologist_Count
FROM Doctor
WHERE specialization ILIKE 'Neurologist';
```

OUTPUT:-

```
 neurologist_count
------------------
        2
```

---

iii) Give the list of all patients who are suffering from “Fever”.

```
SELECT * FROM Patient
WHERE disease ILIKE 'Fever';
```

OUTPUT:-

```
 pno |  pat_name     |  city   | disease
-----+---------------+---------+----------
  1  | Ravi Shinde   | Pune    | Fever
  3  | Rutuja Patil  | Mumbai  | Fever
  5  | Manasi Desai  | Alandi  | Fever
```

---

iv) Find the specialization and phone numbers of all doctors from Alandi.

```
SELECT dname, specialization, phone_no
FROM Doctor
WHERE addr ILIKE 'Alandi';
```

OUTPUT:-

```
   dname         | specialization | phone_no
--------------------+----------------+------------
 Manoj Deshmukh     | Neurologist    | 9876543210
 Ramesh Kulkarni    | Orthopedic     | 9754213698
```

B) Write a stored function using cursors to display all the details of all Patients from Nashik city. [10 Marks]

Solution:

```
-- Create a stored function using cursor
CREATE OR REPLACE FUNCTION get_nashik_patients()
RETURNS TABLE(patient_id INT, patient_name VARCHAR, age INT, city VARCHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    pat_cursor CURSOR FOR
        SELECT patient_id, patient_name, age, city
        FROM Patient
        WHERE city = 'Nashik';
    pat_record RECORD;
BEGIN
    OPEN pat_cursor;
    LOOP
        FETCH pat_cursor INTO pat_record;
        EXIT WHEN NOT FOUND;
        patient_id := pat_record.patient_id;
        patient_name := pat_record.patient_name;
        age := pat_record.age;
        city := pat_record.city;
        RETURN NEXT;
    END LOOP;
    CLOSE pat_cursor;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_nashik_patients();
```

Output (O/P) Example:

```
 patient_id | patient_name | age | city
------------+--------------+-----+------
 101        | Rajesh       | 40  | Nashik
 105        | Priya        | 35  | Nashik
```

I/P: City = `Nashik`
O/P: All patient details from Nashik

Q.2) Write a program to simulate Non-Pre-emptive Shortest Job First (SJF) scheduling. Accept number of processes, arrival time, and burst time. Calculate and display the average waiting time. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, j, temp;
   int at[20], bt[20], wt[20], tat[20], p[20];
   float avg_wait = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
      scanf("%d%d", &at[i], &bt[i]);
      p[i] = i + 1;
   }


   // Sort by Burst Time (Shortest Job First)
   for (i = 0; i < n-1; i++) {
      for (j = i+1; j < n; j++) {
         if (bt[i] > bt[j]) {
            temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
            temp = at[i]; at[i] = at[j]; at[j] = temp;
            temp = p[i]; p[i] = p[j]; p[j] = temp;
         }
      }
```

```
    }

    wt[0] = 0;
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        wt[i] -= at[i];
        avg_wait += wt[i];
    }

    printf("\nProcess\tAT\tBT\tWT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], wt[i]);
    }

    printf("\nAverage Waiting Time = %.2f\n", avg_wait/n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 6
Enter Arrival Time and Burst Time for Process 2: 1 8
Enter Arrival Time and Burst Time for Process 3: 2 7

Process AT      BT      WT
P1      0       6       0
P3      2       7       4
P2      1       8       9
Average Waiting Time = 4.33
```

Assignment 6

Q.1) Consider the following database:

Student (rno, name, city)

Teacher (tno, tname, phone_no, salary)

The relationship is **Student–Teacher: many-to-many**,
with **subject** as a descriptive attribute.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE SchoolDB;

\c SchoolDB;

-- Create Student Table
CREATE TABLE Student (
    rno SERIAL PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);

-- Create Teacher Table
CREATE TABLE Teacher (
    tno SERIAL PRIMARY KEY,
    tname VARCHAR(50),
    phone_no VARCHAR(15),
    salary NUMERIC(10,2)
);

-- Create Relationship Table (Many-to-Many with Descriptive Attribute)
CREATE TABLE Student_Teacher (
```

```
    rno INT REFERENCES Student(rno),
    tno INT REFERENCES Teacher(tno),
    subject VARCHAR(50),
    PRIMARY KEY (rno, tno)
);

-- Insert Records into Student
INSERT INTO Student (name, city)
VALUES
('Shivani Patil', 'Pune'),
('Shubham Deshmukh', 'Nashik'),
('Anjali Kulkarni', 'Mumbai'),
('Shantanu Joshi', 'Pune'),
('Rahul Pawar', 'Nashik');

-- Insert Records into Teacher
INSERT INTO Teacher (tname, phone_no, salary)
VALUES
('Prof. Satkar', '9876543210', 65000),
('Prof. Mehta', '9823456789', 70000),
('Prof. Rane', '9754213698', 55000),
('Prof. Desai', '9123456780', 80000),
('Prof. Pawar', '9988776655', 60000);

-- Insert Records into Student_Teacher
INSERT INTO Student_Teacher (rno, tno, subject)
VALUES
(1, 1, 'DBMS'),
(2, 2, 'OS'),
(3, 3, 'CN'),
(4, 1, 'DBMS'),
```

(5, 4, 'Java');

---

i) List all students whose name starts from ‘Sh’.

SELECT * FROM Student

WHERE name ILIKE 'Sh%';

OUTPUT:-

```
 rno |      name         |  city
-----+-------------------+---------
  1  | Shivani Patil     | Pune
  2  | Shubham Deshmukh  | Nashik
  4  | Shantanu Joshi    | Pune
```

---

ii) Display the count of students from city ‘Pune’.

SELECT COUNT(*) AS Student_Count

FROM Student

WHERE city = 'Pune';

OUTPUT:-

```
 student_count
---------------
      2
```

---

iii) Find the maximum salary of teachers.

SELECT MAX(salary) AS Max_Salary FROM Teacher;

OUTPUT:-

```
 max_salary
------------
   80000.00
```

---

iv) Change the phone number of “Prof. Satkar” to “9822131226”.

UPDATE Teacher

SET phone_no = '9822131226'

WHERE tname = 'Prof. Satkar';

OUTPUT:-

UPDATE 1

(Phone number updated successfully)

B) Create a stored procedure named as “updaterecords” to give 5% rise in salary of teacher. [10 Marks]

Solution:

```
-- Create a stored procedure to update teacher salary by 5%
CREATE OR REPLACE PROCEDURE updaterecords()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Teacher
    SET salary = salary + (salary * 0.05);
END;
$$;
```

Call the Procedure (I/P):

```
CALL updaterecords();
```

Output (O/P):

```
Query OK, X rows affected
```

I/P: N/A (procedure updates all teacher salaries)
O/P: Salary of all teachers increased by 5%

Q.2) Write a program to simulate Non-Preemptive Shortest Job First (SJF) scheduling.
Accept number of processes, arrival time and burst time.
Calculate and display the average turnaround time. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, j, temp;
   int at[20], bt[20], tat[20], wt[20], p[20];
```

```
float avg_tat = 0;

printf("Enter number of processes: ");
scanf("%d", &n);

for (i = 0; i < n; i++) {
   printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
   scanf("%d%d", &at[i], &bt[i]);
   p[i] = i + 1;
}

// Sort by Burst Time
for (i = 0; i < n-1; i++) {
   for (j = i+1; j < n; j++) {
      if (bt[i] > bt[j]) {
         temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
         temp = at[i]; at[i] = at[j]; at[j] = temp;
         temp = p[i]; p[i] = p[j]; p[j] = temp;
      }
   }
}

tat[0] = bt[0] - at[0];
for (i = 1; i < n; i++) {
   tat[i] = tat[i-1] + bt[i] - at[i];
   avg_tat += tat[i];
}

printf("\nProcess\tAT\tBT\tTAT\n");
for (i = 0; i < n; i++) {
   printf("P%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], tat[i]);
```

```
    }

    printf("\nAverage Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 3
Enter Arrival Time and Burst Time for Process 3: 2 8

Process AT      BT      TAT
P1      0       5       5
P2      1       3       7
P3      2       8       15
Average Turnaround Time = 9.00
```

Assignment 7

Q.1) Consider the following database:

Policy (pno, pname, premium_amt, policy_type)

Customer (cno, cname, city, agent_name)

The relationship is **Policy–Customer: many-to-one.**
The policy_type can have values as **'Yearly'**, **'Half-yearly'**, or **'Monthly'**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE InsuranceDB;


\c InsuranceDB;


-- Create Policy Table
CREATE TABLE Policy (
    pno SERIAL PRIMARY KEY,
    pname VARCHAR(50),
    premium_amt NUMERIC(10,2),
    policy_type VARCHAR(20) CHECK (policy_type IN ('Yearly', 'Half-yearly', 'Monthly'))
);


-- Create Customer Table
CREATE TABLE Customer (
    cno SERIAL PRIMARY KEY,
    cname VARCHAR(50),
    city VARCHAR(50),
    agent_name VARCHAR(50),
    pno INT REFERENCES Policy(pno)
);
```

```
-- Insert Records into Policy
INSERT INTO Policy (pname, premium_amt, policy_type)
VALUES
('Life Secure', 12000, 'Yearly'),
('Health Plus', 6000, 'Half-yearly'),
('Safe Future', 1000, 'Monthly'),
('Wealth Builder', 8000, 'Yearly'),
('Smart Health', 1500, 'Monthly');

-- Insert Records into Customer
INSERT INTO Customer (cname, city, agent_name, pno)
VALUES
('Rohit Patil', 'Pune', 'Anita Deshmukh', 1),
('Sneha Mehta', 'Nashik', 'Karan Joshi', 2),
('Ramesh Pawar', 'Pune', 'Anita Deshmukh', 3),
('Anjali Kulkarni', 'Mumbai', 'Suresh Patil', 4),
('Nilesh Jadhav', 'Pune', 'Anita Deshmukh', 5);
```

---

i) List the details of all customers who live in city 'Pune'.

```
SELECT * FROM Customer
WHERE city = 'Pune';
```

OUTPUT:-

```
 cno |   cname       |  city  |   agent_name     | pno
-----+---------------+--------+------------------+-----
  1  | Rohit Patil   | Pune   | Anita Deshmukh   | 1
  3  | Ramesh Pawar  | Pune   | Anita Deshmukh   | 3
  5  | Nilesh Jadhav | Pune   | Anita Deshmukh   | 5
```

---

ii) Display the average premium amount.

```
SELECT AVG(premium_amt) AS Average_Premium
FROM Policy;
```

OUTPUT:-

```
 average_premium
-----------------
    7700.00
```

---

iii) Increase the premium amount for Monthly policies by 10%.

```
UPDATE Policy
SET premium_amt = premium_amt * 1.10
WHERE policy_type = 'Monthly';
```

OUTPUT:-

UPDATE 2

(Premium for Monthly policies increased by 10%)

---

iv) Display the policy type wise count of policies.

```
SELECT policy_type, COUNT(*) AS Policy_Count
FROM Policy
GROUP BY policy_type;
```

OUTPUT:-

```
 policy_type  | policy_count
--------------+--------------
 Yearly       |     2
 Half-yearly  |     1
 Monthly      |     2
```

B) Create a stored function named “max_premium” which will find the maximum premium amount. [10 Marks]

Solution:

```
-- Create a stored function to find maximum premium
CREATE OR REPLACE FUNCTION max_premium()
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    max_amt NUMERIC;
```

```
BEGIN
    SELECT MAX(premium) INTO max_amt
    FROM Policy;  -- Assuming the table storing premium is named 'Policy'

    RETURN max_amt;
END;
$$;
```

Call the Function (I/P):

```
SELECT max_premium();
```

Output (O/P) Example:

```
 max_premium
-------------
 120000
```

I/P: N/A (function calculates max from table)
O/P: Maximum premium amount from the `Policy` table

Q.2) Write a program for Round Robin (RR) scheduling for a given time quantum.
Accept number of processes, arrival time, burst time, and time quantum.
Calculate the waiting time for every process and display the average waiting time. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, tq, time = 0, bt[20], rem_bt[20], at[20], wt[20] = {0}, done = 0;
   float avg_wt = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
      scanf("%d%d", &at[i], &bt[i]);
      rem_bt[i] = bt[i];
   }
```

```
printf("Enter Time Quantum: ");
scanf("%d", &tq);

while (1) {
   done = 1;
   for (i = 0; i < n; i++) {
      if (rem_bt[i] > 0) {
         done = 0;
         if (rem_bt[i] > tq) {
            time += tq;
            rem_bt[i] -= tq;
         } else {
            time += rem_bt[i];
            wt[i] = time - bt[i] - at[i];
            rem_bt[i] = 0;
         }
      }
   }
   if (done == 1)
      break;
}

printf("\nProcess\tAT\tBT\tWT\n");
for (i = 0; i < n; i++) {
   printf("P%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], wt[i]);
   avg_wt += wt[i];
}

printf("\nAverage Waiting Time = %.2f\n", avg_wt / n);
return 0;
```

}

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Arrival Time and Burst Time for Process 1: 0 5

Enter Arrival Time and Burst Time for Process 2: 1 4

Enter Arrival Time and Burst Time for Process 3: 2 6

Enter Time Quantum: 2

Process	AT	BT	WT
P1	0	5	4
P2	1	4	5
P3	2	6	7

Average Waiting Time = 5.33

Assignment 8

Q.1)

Consider the following database:

Item (item_no, name, quantity)

Supplier (s_no, name, city)

The relationship is **Item–Supplier: many-to-many.**

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE StoreDB;

\c StoreDB;

-- Create Item Table
CREATE TABLE Item (
    item_no SERIAL PRIMARY KEY,
    name VARCHAR(50),
    quantity INT
);

-- Create Supplier Table
CREATE TABLE Supplier (
    s_no SERIAL PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);

-- Create Relationship Table (Many-to-Many)
CREATE TABLE Item_Supplier (
    item_no INT REFERENCES Item(item_no),
```

```
    s_no INT REFERENCES Supplier(s_no),
    PRIMARY KEY (item_no, s_no)
);

-- Insert Records into Item
INSERT INTO Item (name, quantity)
VALUES
('Mouse', 500),
('Keyboard', 300),
('Monitor', 200),
('Printer', 150),
('CPU', 100);

-- Insert Records into Supplier
INSERT INTO Supplier (name, city)
VALUES
('Mohan Traders', 'Pune'),
('Mehta Electronics', 'Mumbai'),
('Raj Suppliers', 'Nashik'),
('Manoj Hardware', 'Pune'),
('Global Systems', 'Delhi');

-- Insert Records into Item_Supplier (relationship)
INSERT INTO Item_Supplier (item_no, s_no)
VALUES
(1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 1);
```

---

i) Change the quantity for item “Mouse” to 800.

```
UPDATE Item
SET quantity = 800
WHERE name = 'Mouse';
```

OUTPUT:-

UPDATE 1

(Quantity of 'Mouse' updated successfully)

---

ii) List the details of the suppliers whose name begins with the alphabet "M".

```
SELECT * FROM Supplier
WHERE name ILIKE 'M%';
```

OUTPUT:-

```
 s_no |       name         |  city
------+--------------------+---------
  1   | Mohan Traders      | Pune
  2   | Mehta Electronics  | Mumbai
  4   | Manoj Hardware     | Pune
```

---

iii) Display the count of items.

```
SELECT COUNT(*) AS Item_Count FROM Item;
```

OUTPUT:-

```
 item_count
------------
     5
```

---

iv) List the names of suppliers who do not live in city 'Pune'.

```
SELECT name, city FROM Supplier
WHERE city <> 'Pune';
```

OUTPUT:-

```
     name          |   city
--------------------+-----------
 Mehta Electronics   | Mumbai
 Raj Suppliers       | Nashik
 Global Systems      | Delhi
```

B) Write a stored function to find the minimum quantity of item. [10 Marks]

PL/pgSQL Function:

```
CREATE OR REPLACE FUNCTION min_quantity()
RETURNS INT AS $$
DECLARE
   min_qty INT;
BEGIN
   SELECT MIN(quantity) INTO min_qty FROM Item;
   RETURN min_qty;
END;
$$ LANGUAGE plpgsql;

-- Execute Function
SELECT min_quantity() AS Minimum_Quantity;
```

OUTPUT:-

```
 minimum_quantity
------------------
      100
```

Q.2) Write a program to implement Banker's Algorithm. Mention number of processes and available resources.
Calculate need matrix based on max and allocation matrix. [10 Marks]

C Program:

```
#include <stdio.h>

int main() {
   int n, m, i, j, k;
   int alloc[10][10], max[10][10], avail[10];
   int need[10][10], finish[10] = {0}, safeSeq[10];
   int count = 0;

   printf("Enter number of processes: ");
```

```
scanf("%d", &n);
printf("Enter number of resources: ");
scanf("%d", &m);

printf("Enter Allocation Matrix:\n");
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      scanf("%d", &alloc[i][j]);

printf("Enter Max Matrix:\n");
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      scanf("%d", &max[i][j]);

printf("Enter Available Resources:\n");
for (i = 0; i < m; i++)
   scanf("%d", &avail[i]);

// Calculate Need Matrix
printf("\nNeed Matrix:\n");
for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++) {
      need[i][j] = max[i][j] - alloc[i][j];
      printf("%d ", need[i][j]);
   }
   printf("\n");
}

// Banker's Safety Algorithm
while (count < n) {
   int found = 0;
```

```
    for (i = 0; i < n; i++) {
      if (finish[i] == 0) {
        int flag = 0;
        for (j = 0; j < m; j++) {
          if (need[i][j] > avail[j]) {
            flag = 1;
            break;
          }
        }

        if (flag == 0) {
          for (k = 0; k < m; k++)
            avail[k] += alloc[i][k];
          safeSeq[count++] = i;
          finish[i] = 1;
          found = 1;
        }
      }
    }
    if (found == 0) {
      printf("\nSystem is not in a safe state.\n");
      return 0;
    }
  }

  printf("\nSystem is in a safe state.\nSafe sequence is: ");
  for (i = 0; i < n; i++)
    printf("P%d ", safeSeq[i]);

  return 0;
}
```

SAMPLE INPUT / OUTPUT:-

Enter number of processes: 3

Enter number of resources: 3

Enter Allocation Matrix:

0 1 0

2 0 0

3 0 2

Enter Max Matrix:

7 5 3

3 2 2

9 0 2

Enter Available Resources:

3 3 2

Need Matrix:

7 4 3

1 2 2

6 0 0

System is in a safe state.

Safe sequence is: P1 P0 P2

Assignment 9

Q.1)

Consider the following database:

Student (sno, s_name, s_class)

Teacher (tno, t_name, yrs_experience)

s_class can have values **'FY'**, **'SY'**, or **'TY'**.
The relationship is **Student–Teacher: many-to-many** with **subject** as a descriptive attribute.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE CollegeDB;


\c CollegeDB;


-- Create Student Table
CREATE TABLE Student (
   sno SERIAL PRIMARY KEY,
   s_name VARCHAR(50),
   s_class VARCHAR(10) CHECK (s_class IN ('FY', 'SY', 'TY'))
);


-- Create Teacher Table
CREATE TABLE Teacher (
   tno SERIAL PRIMARY KEY,
   t_name VARCHAR(50),
   yrs_experience INT
);


-- Create Relationship Table (Many-to-Many with descriptive attribute)
CREATE TABLE Student_Teacher (
```

```
    sno INT REFERENCES Student(sno),
    tno INT REFERENCES Teacher(tno),
    subject VARCHAR(50),
    PRIMARY KEY (sno, tno)
);

-- Insert Records into Student
INSERT INTO Student (s_name, s_class)
VALUES
('Rohit Patil', 'FY'),
('Sneha Mehta', 'SY'),
('Anjali Kulkarni', 'TY'),
('Suresh Pawar', 'TY'),
('Rutuja Deshmukh', 'FY'),
('Amit Joshi', 'SY');

-- Insert Records into Teacher
INSERT INTO Teacher (t_name, yrs_experience)
VALUES
('Prof. Desai', 12),
('Prof. Kulkarni', 8),
('Prof. Sharma', 15),
('Prof. Mehta', 10);

-- Insert Records into Student_Teacher
INSERT INTO Student_Teacher (sno, tno, subject)
VALUES
(1, 1, 'DBMS'),
(2, 2, 'OS'),
(3, 3, 'Networking'),
(4, 1, 'DBMS'),
```

(5, 4, 'C Programming'),

(6, 3, 'OS');

---

i) Give class-wise number of students.

```
SELECT s_class, COUNT(*) AS Student_Count
FROM Student
GROUP BY s_class;
```

OUTPUT:-

```
 s_class | student_count
---------+---------------
 FY      |     2
 SY      |     2
 TY      |     2
```

---

ii) List all students studying in class “TY”.

```
SELECT * FROM Student
WHERE s_class = 'TY';
```

OUTPUT:-

```
 sno |     s_name       | s_class
-----+------------------+---------
  3  | Anjali Kulkarni  | TY
  4  | Suresh Pawar     | TY
```

---

iii) Count the number of students who have taken subject “OS”.

```
SELECT COUNT(*) AS OS_Students
FROM Student_Teacher
WHERE subject ILIKE 'OS';
```

OUTPUT:-

```
 os_students
-------------
      2
```

---

iv) Delete record of student whose sno = 101.

DELETE FROM Student

WHERE sno = 101;

OUTPUT:-

DELETE 0

(No record found for sno = 101 or record deleted successfully)

B) Write a stored function to take teacher name as input and return the years of experience of that teacher. [10 Marks]

Solution:

```
-- Create a stored function to return years of experience of a teacher
CREATE OR REPLACE FUNCTION get_experience(p_name VARCHAR)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    years_exp INT;
BEGIN
    SELECT experience INTO years_exp
    FROM Teacher
    WHERE teacher_name = p_name;

    RETURN years_exp;
END;
$$;
```

Call the Function (I/P):

```
SELECT get_experience('Shivraj');
```

Output (O/P) Example:

```
 get_experience
----------------
       10
```

I/P: `'Shivraj'`
O/P: `10` (years of experience of Shivraj)

Q.2) Consider a system with 'm' processes and 'n' resource types.
Accept number of instances for each resource type.

For each process, accept the allocation and maximum requirement matrices.
Write a program to display the contents of the need matrix. [10 Marks]

C Program:

```
#include <stdio.h>
int main() {
   int n, m, i, j;
   int alloc[10][10], max[10][10], need[10][10];
   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter number of resource types: ");
   scanf("%d", &m);
   printf("Enter Allocation Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         scanf("%d", &alloc[i][j]);
      }
   }
   printf("Enter Maximum Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         scanf("%d", &max[i][j]);
      }
   }
   // Calculate Need Matrix
   printf("\nNeed Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         need[i][j] = max[i][j] - alloc[i][j];
         printf("%d ", need[i][j]);
      }
      printf("\n");
```

```
    }

    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter number of resource types: 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
Enter Maximum Matrix:
7 5 3
3 2 2
9 0 2

Need Matrix:
7 4 3
1 2 2
6 0 0
```

Assignment 10

Q.1) Consider the following database:

Account (acct_no, acct_type, balance, branch_name)

Customer (cust_no, cust_name, cust_city)

The relationship is **Customer–Account: 1–M (one-to-many)**.
acct_type can be either **"saving"** or **"current"**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE BankDB;

\c BankDB;

-- Create Account Table
CREATE TABLE Account (
    acct_no SERIAL PRIMARY KEY,
    acct_type VARCHAR(10) CHECK (acct_type IN ('saving', 'current')),
    balance NUMERIC(12,2),
    branch_name VARCHAR(50),
    cust_no INT
);

-- Create Customer Table
CREATE TABLE Customer (
    cust_no SERIAL PRIMARY KEY,
    cust_name VARCHAR(50),
    cust_city VARCHAR(50)
);
```

```
-- Add foreign key relationship
ALTER TABLE Account
ADD CONSTRAINT fk_customer FOREIGN KEY (cust_no) REFERENCES Customer(cust_no);

-- Insert Records into Customer
INSERT INTO Customer (cust_name, cust_city)
VALUES
('Rohit Mehta', 'Pune'),
('Sneha Desai', 'Mumbai'),
('Ramesh Pawar', 'Nashik'),
('Anjali Kulkarni', 'Pune'),
('Rajesh Patil', 'Pune');

-- Insert Records into Account
INSERT INTO Account (acct_type, balance, branch_name, cust_no)
VALUES
('saving', 750000, 'M.G.Road', 1),
('current', 350000, 'Camp', 2),
('saving', 250000, 'JM Road', 3),
('saving', 800000, 'M.G.Road', 4),
('current', 100000, 'Karve Nagar', 5);
```

---

i) Display information of all saving accounts having balance > 500,000.

```
SELECT * FROM Account
WHERE acct_type = 'saving' AND balance > 500000;
```

OUTPUT:-

```
 acct_no | acct_type |  balance  | branch_name | cust_no
---------+-----------+-----------+-------------+---------
   1    | saving    | 750000.00 | M.G.Road    | 1
   4    | saving    | 800000.00 | M.G.Road    | 4
```

---

ii) Count customers whose name starts with ‘R’.

```
SELECT COUNT(*) AS Customer_Count
FROM Customer
WHERE cust_name ILIKE 'R%';
```

OUTPUT:-

```
 customer_count
----------------
      3
```

---

iii) Find the total balance at branch “M.G.Road”.

```
SELECT SUM(balance) AS Total_Balance
FROM Account
WHERE branch_name = 'M.G.Road';
```

OUTPUT:-

```
 total_balance
---------------
  1550000.00
```

---

iv) Delete the record whose cust_name is “Ramesh Pawar”.

```
DELETE FROM Customer
WHERE cust_name = 'Ramesh Pawar';
```

OUTPUT:-

```
DELETE 1
(Customer record deleted successfully)
```

B) Write a stored function using cursors to print names of all customers from a given city. [10 Marks]

Solution:

```
-- Create a stored function using cursor to get customer names from a given city
CREATE OR REPLACE FUNCTION get_customers_by_city(p_city VARCHAR)
RETURNS TABLE(customer_name VARCHAR)
LANGUAGE plpgsql
AS $$
```

```
DECLARE
    cust_cursor CURSOR FOR
        SELECT customer_name
        FROM Customer
        WHERE city = p_city;
    cust_record RECORD;
BEGIN
    OPEN cust_cursor;
    LOOP
        FETCH cust_cursor INTO cust_record;
        EXIT WHEN NOT FOUND;
        customer_name := cust_record.customer_name;
        RETURN NEXT;
    END LOOP;
    CLOSE cust_cursor;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_customers_by_city('Pune');
```

Output (O/P) Example:

```
 customer_name
---------------
 Rajesh
 Priya
 Anil
```

I/P: `'Pune'`
O/P: Names of all customers from Pune

Q.2) Write a Program to implement following functionality:

Accept the **Available**, **Allocation**, and **Max** matrices for given processes and resources, and **display the contents of the Need matrix.**

Process	Allocation	Max	Available
	A B C	A B C	A B C
P1	0 1 0	7 5 3	3 3 2
P2	2 0 0	3 2 2	
P3	3 0 2	9 0 2	
P4	2 1 1	2 2 2	
P5	0 0 2	4 3 3	

C Program:

```
#include <stdio.h>

int main() {
```

```
int n, m, i, j;
int alloc[10][10], max[10][10], need[10][10], avail[10];

printf("Enter number of processes: ");
scanf("%d", &n);
printf("Enter number of resources: ");
scanf("%d", &m);

printf("Enter Allocation Matrix:\n");
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      scanf("%d", &alloc[i][j]);

printf("Enter Max Matrix:\n");
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      scanf("%d", &max[i][j]);

printf("Enter Available Resources:\n");
for (i = 0; i < m; i++)
   scanf("%d", &avail[i]);

printf("\nNeed Matrix:\n");
for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++) {
      need[i][j] = max[i][j] - alloc[i][j];
      printf("%d ", need[i][j]);
   }
   printf("\n");
}
```

```
    return 0;
}
```

SAMPLE INPUT / OUTPUT:-

Enter number of processes: 5

Enter number of resources: 3

Enter Allocation Matrix:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter Max Matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Available Resources:

3 3 2

Need Matrix:

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

Assignment 11

Q.1) Consider the following database:

Bus (Bus_no, capacity, depot_name)

Route (Route_no, source, destination, no_of_stations)

Relationship: Bus–Route : M–1 (Many-to-One)
Constraint: Bus capacity is **NOT NULL**

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE TransportDB;

\c TransportDB;

-- Create Route Table
CREATE TABLE Route (
    route_no SERIAL PRIMARY KEY,
    source VARCHAR(50),
    destination VARCHAR(50),
    no_of_stations INT
);

-- Create Bus Table
CREATE TABLE Bus (
    bus_no SERIAL PRIMARY KEY,
    capacity INT NOT NULL,
    depot_name VARCHAR(50),
    route_no INT REFERENCES Route(route_no)
);

-- Insert Records into Route
```

```
INSERT INTO Route (source, destination, no_of_stations)
VALUES
('Pune', 'Mumbai', 12),
('Nashik', 'Nagpur', 15),
('Pune', 'Satara', 8),
('Kolhapur', 'Sangli', 6),
('Pune', 'Nashik', 10);

-- Insert Records into Bus
INSERT INTO Bus (capacity, depot_name, route_no)
VALUES
(50, 'Swargate', 1),
(45, 'Shivajinagar', 2),
(60, 'Swargate', 3),
(40, 'Nigdi', 4),
(55, 'Swargate', 5);
```

---

i) List all buses which belong to depot 'Swargate'.

```
SELECT * FROM Bus
WHERE depot_name = 'Swargate';
```

OUTPUT:-

```
 bus_no | capacity | depot_name | route_no
--------+----------+-------------+----------
  1     |   50     | Swargate   | 1
  3     |   60     | Swargate   | 3
  5     |   55     | Swargate   | 5
```

---

ii) Delete Bus details whose Bus number is 2.

```
DELETE FROM Bus
WHERE bus_no = 2;
```

OUTPUT:-

DELETE 1

(Bus record deleted successfully)

---

iii) List the route details having number of stations > 10.

SELECT * FROM Route

WHERE no_of_stations > 10;

OUTPUT:-

```
 route_no |  source  | destination | no_of_stations
----------+----------+-------------+----------------
  1       | Pune     | Mumbai      | 12
  2       | Nashik   | Nagpur      | 15
```

---

iv) List all routes starting from Station 'Pune'.

SELECT * FROM Route

WHERE source = 'Pune';

OUTPUT:-

```
 route_no | source | destination | no_of_stations
----------+--------+-------------+----------------
  1       | Pune   | Mumbai      | 12
  3       | Pune   | Satara      | 8
  5       | Pune   | Nashik      | 10
```

B) Write a stored function using cursors to accept `route_no` from the user and display the number of stations of that route. [10 Marks]

Solution:

```
-- Create a stored function using cursor to count number of stations for a given
route
CREATE OR REPLACE FUNCTION count_stations(p_route_no INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    station_cursor CURSOR FOR
        SELECT station_id
        FROM Route_Stations
        WHERE route_no = p_route_no;
```

```
    station_record RECORD;
    station_count INT := 0;
BEGIN
    OPEN station_cursor;
    LOOP
        FETCH station_cursor INTO station_record;
        EXIT WHEN NOT FOUND;
        station_count := station_count + 1;
    END LOOP;
    CLOSE station_cursor;

    RETURN station_count;
END;
$$;
```

Call the Function (I/P):

```
SELECT count_stations(101);
```

Output (O/P) Example:

```
 count_stations
----------------
       12
```

I/P: `101` (route number)
O/P: `12` (number of stations in route 101)

Q.2) Write a program to simulate Non-Preemptive Shortest Job First (SJF) Scheduling.
Accept number of processes, arrival time, and burst time from the user.
Display waiting time for each process and calculate the **average waiting time.** [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, j, pos, temp;
   int bt[20], at[20], wt[20], tat[20], p[20];
   float avg_wt = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);
```

```
for (i = 0; i < n; i++) {
    printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
    scanf("%d%d", &at[i], &bt[i]);
    p[i] = i + 1;
}

// Sort according to Burst Time
for (i = 0; i < n; i++) {
    pos = i;
    for (j = i + 1; j < n; j++) {
        if (bt[j] < bt[pos])
            pos = j;
    }
    temp = bt[i]; bt[i] = bt[pos]; bt[pos] = temp;
    temp = at[i]; at[i] = at[pos]; at[pos] = temp;
    temp = p[i]; p[i] = p[pos]; p[pos] = temp;
}

wt[0] = 0;
for (i = 1; i < n; i++) {
    wt[i] = 0;
    for (j = 0; j < i; j++)
        wt[i] += bt[j];
    wt[i] -= at[i];
    avg_wt += wt[i];
}

printf("\nProcess\tAT\tBT\tWT\n");
for (i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], wt[i]);
```

```
    }

    printf("\nAverage Waiting Time = %.2f\n", avg_wt / n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 3
Enter Arrival Time and Burst Time for Process 3: 2 8

Process AT      BT      WT
P2      1       3       0
P1      0       5       3
P3      2       8       8

Average Waiting Time = 3.67
```

Assignment 12

Q.1) Consider the following database:

Game (gcode, gname, noofplayers, coachname, captain_name)

Player (pno, pname)

There exists a **one-to-many** relationship between **Game** and **Player**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE SportsDB;

\c SportsDB;

-- Create Game Table
CREATE TABLE Game (
    gcode SERIAL PRIMARY KEY,
    gname VARCHAR(50),
    noofplayers INT,
    coachname VARCHAR(50),
    captain_name VARCHAR(50)
);

-- Create Player Table
CREATE TABLE Player (
    pno SERIAL PRIMARY KEY,
    pname VARCHAR(50),
    gcode INT REFERENCES Game(gcode)
);

-- Insert Records into Game
INSERT INTO Game (gname, noofplayers, coachname, captain_name)
```

```
VALUES
('Cricket', 11, 'Rahul Dravid', 'Rohit Sharma'),
('Football', 11, 'Igor Stimac', 'Sunil Chhetri'),
('Hockey', 11, 'Harendra Singh', 'Manpreet Singh'),
('Kho Kho', 12, 'Ashok Kumar', 'Suresh Patil'),
('Basketball', 5, 'David Williams', 'Rahul Mehta');

-- Insert Records into Player
INSERT INTO Player (pname, gcode)
VALUES
('Rohit Sharma', 1),
('Virat Kohli', 1),
('Sunil Chhetri', 2),
('Manpreet Singh', 3),
('Suresh Patil', 4),
('Rahul Mehta', 5);
```

---

i) Display all game names that end with “ball”.

```
SELECT gname FROM Game
WHERE gname ILIKE '%ball';
```

OUTPUT:-

```
  gname
------------
 Football
 Basketball
```

---

ii) Give the average number of players.

```
SELECT AVG(noofplayers) AS Average_No_Of_Players FROM Game;
```

OUTPUT:-

```
 average_no_of_players
-----------------------
```

10.0

---

iii) Display all details of game “Kho Kho”.

SELECT * FROM Game

WHERE gname ILIKE 'Kho Kho';

OUTPUT:-

```
 gcode |  gname  | noofplayers |  coachname  |  captain_name
-------+----------+-------------+--------------+---------------
   4   | Kho Kho |     12      | Ashok Kumar  | Suresh Patil
```

---

iv) Update the coach name from “Harendra Singh” to “Rajesh Sharma” for game “Hockey”.

UPDATE Game

SET coachname = 'Rajesh Sharma'

WHERE gname = 'Hockey' AND coachname = 'Harendra Singh';

OUTPUT:-

UPDATE 1

(Coach name updated successfully)

B) Create a stored procedure named “deleterecords” for deleting the Game record having a specific coach name. [10 Marks]

Solution:

```
-- Create a stored procedure to delete game records by coach name
CREATE OR REPLACE PROCEDURE deleterecords(p_coach_name VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM Game
    WHERE coach_name = p_coach_name;
END;
$$;
```

Call the Procedure (I/P):

```
CALL deleterecords('Shivraj');
```

Output (O/P):

```
Query OK, X rows affected
```

I/P: `'Shivraj'`
O/P: All records of games having coach name 'Shivraj' are deleted

Q.2)

Write a program to simulate Non-preemptive Shortest Job First (SJF) CPU scheduling.
Accept number of processes, arrival time, and burst time from the user.
Display **turnaround time (TAT)** for each process and calculate **average turnaround time**. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, j, pos, temp;
   int at[20], bt[20], tat[20], p[20];
   float avg_tat = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
      scanf("%d%d", &at[i], &bt[i]);
      p[i] = i + 1;
   }


   // Sort by Burst Time (Shortest Job First)
   for (i = 0; i < n; i++) {
      pos = i;
      for (j = i + 1; j < n; j++) {
         if (bt[j] < bt[pos])
            pos = j;
```

```
        }

        // Swap
        temp = bt[i]; bt[i] = bt[pos]; bt[pos] = temp;
        temp = at[i]; at[i] = at[pos]; at[pos] = temp;
        temp = p[i]; p[i] = p[pos]; p[pos] = temp;
    }

    tat[0] = bt[0];
    for (i = 1; i < n; i++) {
        tat[i] = tat[i - 1] + bt[i];
        avg_tat += tat[i];
    }

    printf("\nProcess\tAT\tBT\tTAT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], tat[i]);
    }

    printf("\nAverage Turnaround Time = %.2f\n", avg_tat / n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 6
Enter Arrival Time and Burst Time for Process 2: 1 8
Enter Arrival Time and Burst Time for Process 3: 2 7

Process AT      BT      TAT
P1      0       6       6
```

P3	2	7	13
P2	1	8	21

Average Turnaround Time = 13.33

Assignment 13

Q.1) Consider the following database:

Item (item_no, name, quantity, rate)

Supplier (s_no, name, city, contact)

The relationship is **Item–Supplier: many-to-many**.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE StoreDB;

\c StoreDB;

-- Create Item Table
CREATE TABLE Item (
    item_no SERIAL PRIMARY KEY,
    name VARCHAR(50),
    quantity INT,
    rate NUMERIC(10,2)
);

-- Create Supplier Table
CREATE TABLE Supplier (
    s_no SERIAL PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50),
    contact VARCHAR(15)
);

-- Create Relationship Table (Many-to-Many)
CREATE TABLE Item_Supplier (
```

```
    item_no INT REFERENCES Item(item_no),
    s_no INT REFERENCES Supplier(s_no),
    PRIMARY KEY (item_no, s_no)
);

-- Insert Records into Item
INSERT INTO Item (name, quantity, rate)
VALUES
('Pen', 100, 10),
('Pencil', 50, 5),
('Notebook', 40, 60),
('Marker', 25, 80),
('Scale', 35, 20);

-- Insert Records into Supplier
INSERT INTO Supplier (name, city, contact)
VALUES
('Prakash Traders', 'Pune', '9876543210'),
('Mehta Stationers', 'Mumbai', '9823456789'),
('Patil Distributors', 'Pune', '9123456780'),
('Global Supply', 'Delhi', '9754213698');

-- Insert Relationship (Item-Supplier)
INSERT INTO Item_Supplier (item_no, s_no)
VALUES
(1, 1), (2, 2), (3, 3), (4, 1), (5, 4);
```

---

i) List the details of the suppliers whose name begins with the alphabet “P”.

```
SELECT * FROM Supplier
WHERE name ILIKE 'P%';
```

OUTPUT:-

```
 s_no |       name          |  city  |   contact
------+---------------------+--------+-------------
  1   | Prakash Traders      | Pune   | 9876543210
  3   | Patil Distributors   | Pune   | 9123456780
```

---

ii) Delete record of item_no = 4.

```
DELETE FROM Item
WHERE item_no = 4;
```

OUTPUT:-

```
DELETE 1
(Record of item_no 4 deleted successfully)
```

---

iii) Display the count of items with rate > 50Rs.

```
SELECT COUNT(*) AS High_Rate_Items
FROM Item
WHERE rate > 50;
```

OUTPUT:-

```
 high_rate_items
----------------
      1
```

---

iv) List the names of suppliers living in city 'Pune'.

```
SELECT name, contact FROM Supplier
WHERE city = 'Pune';
```

OUTPUT:-

```
       name          |   contact
-------------------------+-------------
 Prakash Traders         | 9876543210
 Patil Distributors      | 9123456780
```

B) Write a function to find the details of items whose quantity is greater than 30. [10 Marks]

Solution:

```
-- Create a stored function to get item details with quantity > 30
CREATE OR REPLACE FUNCTION get_items_high_quantity()
RETURNS TABLE(item_id INT, item_name VARCHAR, quantity INT, price NUMERIC)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT item_id, item_name, quantity, price
    FROM Items
    WHERE quantity > 30;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_items_high_quantity();
```

Output (O/P) Example:

```
 item_id | item_name | quantity | price
---------+-----------+----------+------
 101     | Pen       | 50       | 10
 103     | Notebook  | 40       | 25
```

Q.2) Write a program to simulate FCFS CPU scheduling.
Accept the number of processes, arrival time, and burst time from the user.
Display the **Gantt chart** and **waiting time** for each process, and calculate the **average waiting time**. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {

   int n, i;

   int at[20], bt[20], wt[20], tat[20], ct[20];

   float avg_wt = 0;


   printf("Enter number of processes: ");

   scanf("%d", &n);


   for (i = 0; i < n; i++) {
```

```
    printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
    scanf("%d%d", &at[i], &bt[i]);
}

// First process
ct[0] = at[0] + bt[0];
tat[0] = ct[0] - at[0];
wt[0] = tat[0] - bt[0];

// Remaining processes
for (i = 1; i < n; i++) {
    if (ct[i-1] < at[i])
        ct[i] = at[i] + bt[i];
    else
        ct[i] = ct[i-1] + bt[i];

    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    avg_wt += wt[i];
}

// Display
printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
for (i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
}

printf("\nGantt Chart:\n");
for (i = 0; i < n; i++) {
    printf("|  P%d  ", i+1);
}
```

```
    printf("|\n");

    for (i = 0; i < n; i++) {
        printf("%d\t", ct[i]);
    }

    printf("\n\nAverage Waiting Time = %.2f\n", avg_wt/n);
    return 0;
}
```

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Arrival Time and Burst Time for Process 1: 0 5

Enter Arrival Time and Burst Time for Process 2: 1 3

Enter Arrival Time and Burst Time for Process 3: 2 8

Process	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	8	16	14	6

Gantt Chart:

| P1 | P2 | P3 |

5 8 16

Average Waiting Time = 3.33

Assignment 14

Q.1) Consider the following database:

Book (Book_no, title, author, price, year_published)

Customer (cid, cname, addr)

Relationship: Book–Customer : many-to-many with **quantity** as a descriptive attribute.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE BookStoreDB;

\c BookStoreDB;

-- Create Book Table
CREATE TABLE Book (
    book_no SERIAL PRIMARY KEY,
    title VARCHAR(100),
    author VARCHAR(50),
    price NUMERIC(10,2),
    year_published INT
);

-- Create Customer Table
CREATE TABLE Customer (
    cid SERIAL PRIMARY KEY,
    cname VARCHAR(50),
    addr VARCHAR(100)
);

-- Create Relationship Table (Many-to-Many)
CREATE TABLE Book_Customer (
```

```
    book_no INT REFERENCES Book(book_no),
    cid INT REFERENCES Customer(cid),
    quantity INT,
    PRIMARY KEY (book_no, cid)
);

-- Insert Records into Book
INSERT INTO Book (title, author, price, year_published)
VALUES
('DBMS Fundamentals', 'Korth', 500.00, 2023),
('Operating Systems', 'Galvin', 650.00, 2024),
('Computer Networks', 'Tanenbaum', 800.00, 2022),
('Artificial Intelligence', 'Russell', 950.00, 2024),
('Java Programming', 'Herbert Schildt', 600.00, 2021);

-- Insert Records into Customer
INSERT INTO Customer (cname, addr)
VALUES
('Rohit Patil', 'Pune'),
('Sneha Deshmukh', 'Mumbai'),
('Ramesh Pawar', 'Pune'),
('Anjali Kulkarni', 'Delhi');

-- Insert Records into Book_Customer
INSERT INTO Book_Customer (book_no, cid, quantity)
VALUES
(1, 1, 2),
(2, 2, 1),
(3, 1, 3),
(4, 3, 1),
(5, 4, 2);
```

---

i) Display customer details staying at “Pune”.

```
SELECT * FROM Customer
WHERE addr = 'Pune';
```

OUTPUT:-

```
 cid |    cname       |   addr
-----+----------------+---------
  1  | Rohit Patil    | Pune
  3  | Ramesh Pawar   | Pune
```

---

ii) Display author-wise details of books.

```
SELECT author, COUNT(*) AS No_of_Books, AVG(price) AS Avg_Price
FROM Book
GROUP BY author;
```

OUTPUT:-

```
    author         | no_of_books | avg_price
---------------------+--------------+-----------
 Korth              |     1       | 500.00
 Galvin             |     1       | 650.00
 Tanenbaum          |     1       | 800.00
 Russell            |     1       | 950.00
 Herbert Schildt    |     1       | 600.00
```

---

iii) Display the average price of a book.

```
SELECT AVG(price) AS Average_Book_Price FROM Book;
```

OUTPUT:-

```
 average_book_price
--------------------
       700.00
```

---

iv) Delete the record from Book table with Book_no = 5.

```
DELETE FROM Book

WHERE book_no = 5;
```

OUTPUT:-

```
DELETE 1

(Book record deleted successfully)
```

B) Write a function to define a cursor to print the details of the Books published in year 2024. [10 Marks]

Solution:

```
-- Create a stored function using cursor to get books published in 2024
CREATE OR REPLACE FUNCTION get_books_2024()
RETURNS TABLE(book_id INT, book_name VARCHAR, author VARCHAR, publish_year INT)
LANGUAGE plpgsql
AS $$
DECLARE
    book_cursor CURSOR FOR
        SELECT book_id, book_name, author, publish_year
        FROM Books
        WHERE publish_year = 2024;
    book_record RECORD;
BEGIN
    OPEN book_cursor;
    LOOP
        FETCH book_cursor INTO book_record;
        EXIT WHEN NOT FOUND;
        book_id := book_record.book_id;
        book_name := book_record.book_name;
        author := book_record.author;
        publish_year := book_record.publish_year;
        RETURN NEXT;
    END LOOP;
    CLOSE book_cursor;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_books_2024();
```

Output (O/P) Example:

```
 book_id | book_name       | author      | publish_year
---------+-----------------+-------------+--------------
 101     | SQL Basics      | Rajesh      | 2024
 105     | Advanced PL/pgSQL | Priya     | 2024
```

Q.2) Write a program to simulate FCFS CPU scheduling.
Accept number of processes, arrival time, and burst time from the user.
Display the **Gantt chart** and **turnaround time (TAT)** for each process.
Also find the **average turnaround time**. [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i;
   int at[20], bt[20], tat[20], ct[20];
   float avg_tat = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Arrival Time and Burst Time for Process %d: ", i+1);
      scanf("%d%d", &at[i], &bt[i]);
   }


   // Calculate completion, turnaround times
   ct[0] = at[0] + bt[0];
   tat[0] = ct[0] - at[0];


   for (i = 1; i < n; i++) {
      if (ct[i-1] < at[i])
         ct[i] = at[i] + bt[i];
      else
         ct[i] = ct[i-1] + bt[i];
      tat[i] = ct[i] - at[i];
      avg_tat += tat[i];
```

```
    }

    printf("\nProcess\tAT\tBT\tCT\tTAT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i]);
    }

    printf("\nGantt Chart:\n");
    for (i = 0; i < n; i++) {
        printf("|  P%d  ", i+1);
    }
    printf("|\n");

    for (i = 0; i < n; i++) {
        printf("%d\t", ct[i]);
    }

    printf("\n\nAverage Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 3
Enter Arrival Time and Burst Time for Process 3: 2 8

Process AT      BT      CT      TAT
P1      0       5       5       5
P2      1       3       8       7
P3      2       8       16      14
```

Gantt Chart:

| P1 | P2 | P3 |

5 8 16

Average Turnaround Time = 8.67

Assignment 15

Q.1) Consider the following database:

Sales_order (s_orderno, s_order_date, order_amt)

Client (client_no, name, address)

The relationship is **Client–Sales_order: one-to-many.**
The constraint: **order_amt > 0**

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database

CREATE DATABASE SalesDB;


\c SalesDB;


-- Create Client Table

CREATE TABLE Client (

   client_no SERIAL PRIMARY KEY,

   name VARCHAR(50),

   address VARCHAR(100)

);


-- Create Sales_order Table

CREATE TABLE Sales_order (

   s_orderno SERIAL PRIMARY KEY,

   s_order_date DATE,

   order_amt NUMERIC(10,2) CHECK (order_amt > 0),

   client_no INT REFERENCES Client(client_no)

);


-- Insert Records into Client

INSERT INTO Client (name, address)
```

```
VALUES
('Rohit Patil', 'Pune'),
('Sneha Desai', 'Nasik'),
('Anjali Kulkarni', 'Mumbai'),
('Ramesh Pawar', 'Nasik');

-- Insert Records into Sales_order
INSERT INTO Sales_order (s_order_date, order_amt, client_no)
VALUES
('2024-03-10', 15000, 1),
('2023-12-15', 25000, 2),
('2023-10-05', 18000, 3),
('2024-02-01', 30000, 4);
```

---

i) Display all sale records having order date before “2024-01-01”.

```
SELECT * FROM Sales_order
WHERE s_order_date < '2024-01-01';
```

OUTPUT:-

```
 s_orderno | s_order_date | order_amt | client_no
-----------+--------------+-----------+----------
    2      | 2023-12-15   | 25000.00  |    2
    3      | 2023-10-05   | 18000.00  |    3
```

---

ii) Find maximum sales order amount.

```
SELECT MAX(order_amt) AS Max_Order_Amount FROM Sales_order;
```

OUTPUT:-

```
 max_order_amount
-----------------
     30000.00
```

---

iii) Update the client address of all clients from “Nasik” to “Ahilyanagar”.

UPDATE Client

SET address = 'Ahilyanagar'

WHERE address = 'Nasik';

OUTPUT:-

UPDATE 2

(Client address updated successfully)

---

iv) Add column order_status to the Sales_order table.

ALTER TABLE Sales_order

ADD COLUMN order_status VARCHAR(20);

OUTPUT:-

ALTER TABLE

(Column added successfully)

B) Create a stored procedure named “addrecords” for adding new sales order records. [10 Marks]

Solution:

```
-- Create a stored procedure to add new sales order records
CREATE OR REPLACE PROCEDURE addrecords(
    p_order_id INT,
    p_customer_id INT,
    p_order_date DATE,
    p_amount NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO SalesOrder(order_id, customer_id, order_date, amount)
    VALUES (p_order_id, p_customer_id, p_order_date, p_amount);
END;
$$;
```

Call the Procedure (I/P):

```
CALL addrecords(101, 201, '2025-11-13', 5000);
```

Output (O/P):

```
Query OK, 1 row affected
```

Q.2) Write a program to simulate Pre-emptive Shortest Job First (SJF) Scheduling.
Accept the number of processes, arrival time, and burst time from the user.
Display the **waiting time** for each process and calculate the **average waiting time. [10 Marks]**

C Program:

```
#include <stdio.h>
#include <limits.h>

int main() {
    int n, i, completed = 0, time = 0, smallest;
    float avg_wt = 0;
    int at[20], bt[20], rt[20], wt[20], tat[20], ct[20], min;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
        scanf("%d%d", &at[i], &bt[i]);
        rt[i] = bt[i];
    }

    while (completed != n) {
        smallest = -1;
        min = INT_MAX;

        for (i = 0; i < n; i++) {
            if (at[i] <= time && rt[i] > 0 && rt[i] < min) {
                min = rt[i];
                smallest = i;
            }
        }
```

```
        if (smallest == -1) {
            time++;
            continue;
        }

        rt[smallest]--;
        time++;

        if (rt[smallest] == 0) {
            completed++;
            ct[smallest] = time;
            tat[smallest] = ct[smallest] - at[smallest];
            wt[smallest] = tat[smallest] - bt[smallest];
            avg_wt += wt[smallest];
        }
    }

    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
               i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
    }

    printf("\nAverage Waiting Time = %.2f\n", avg_wt / n);
    return 0;
}
```

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Arrival Time and Burst Time for Process 1: 0 7

Enter Arrival Time and Burst Time for Process 2: 2 4

Enter Arrival Time and Burst Time for Process 3: 4 1

Process	AT	BT	CT	TAT	WT
P1	0	7	12	12	5
P2	2	4	8	6	2
P3	4	1	5	1	0

Average Waiting Time = 2.33

Max. Marks: 35

Assignment 16

Q.1) Consider the following database:

Car (car_code, c_name, c_price, color_type)

Customer (cust_code, cust_name, cust_address)

Constraint: color_type can be either *“metallic”* or *“solid”*
Relationship: Customer–Car : one-to-many

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database

CREATE DATABASE CarDB;


\c CarDB;


-- Create Customer Table

CREATE TABLE Customer (

   cust_code SERIAL PRIMARY KEY,

   cust_name VARCHAR(50),

   cust_address VARCHAR(100)

);


-- Create Car Table

CREATE TABLE Car (

   car_code SERIAL PRIMARY KEY,

   c_name VARCHAR(50),

   c_price NUMERIC(12,2),

   color_type VARCHAR(20) CHECK (color_type IN ('metallic', 'solid')),

   cust_code INT REFERENCES Customer(cust_code)

);


-- Insert Records into Customer
```

```
INSERT INTO Customer (cust_name, cust_address)
VALUES
('Bharat Patil', 'ShivajiNagar'),
('Bhavna Deshmukh', 'Kothrud'),
('Ramesh Pawar', 'ShivajiNagar'),
('Suresh Mehta', 'Pimpri');

-- Insert Records into Car
INSERT INTO Car (c_name, c_price, color_type, cust_code)
VALUES
('Ferrari', 400000, 'metallic', 1),
('BMW', 550000, 'solid', 2),
('Audi', 300000, 'metallic', 3),
('Tata', 180000, 'solid', 4),
('Ferrari', 450000, 'metallic', 2);
```

---

i) Find the names of all Customers whose name start with “B”.

```
SELECT cust_name FROM Customer
WHERE cust_name ILIKE 'B%';
```

OUTPUT:-

```
  cust_name
-------------------
 Bharat Patil
 Bhavna Deshmukh
```

---

ii) Count the number of “metallic” cars.

```
SELECT COUNT(*) AS Metallic_Car_Count
FROM Car
WHERE color_type = 'metallic';
```

OUTPUT:-

```
 metallic_car_count
```

3

---

iii) Give the list of all customers staying in ShivajiNagar.

```
SELECT * FROM Customer
WHERE cust_address = 'ShivajiNagar';
```

OUTPUT:-

```
 cust_code |    cust_name      |  cust_address
-----------+-------------------+---------------
   1       | Bharat Patil      | ShivajiNagar
   3       | Ramesh Pawar      | ShivajiNagar
```

---

iv) Increase the price of all "Ferrari" cars by 15%.

```
UPDATE Car
SET c_price = c_price * 1.15
WHERE c_name = 'Ferrari';
```

OUTPUT:-

```
UPDATE 2
```

(Ferrari car prices increased by 15%)

B) Write a stored function to display details of all metallic coloured cars having price in the range 100,000 to 500,000. [10 Marks]

Solution:

```
-- Create a stored function to get metallic coloured cars within a specific price
range
CREATE OR REPLACE FUNCTION get_metallic_cars()
RETURNS TABLE(car_id INT, car_name VARCHAR, color VARCHAR, price NUMERIC)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT car_id, car_name, color, price
    FROM Cars
    WHERE color ILIKE '%metallic%'
      AND price BETWEEN 100000 AND 500000;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_metallic_cars();
```

Output (O/P) Example:

```
 car_id | car_name    | color       | price
--------+-------------+-------------+--------
 101    | Swift       | Metallic    | 200000
 105    | Creta       | Metallic    | 450000
```

Q.2) Write a program to simulate Pre-emptive Shortest Job First (SJF) Scheduling.
Accept number of processes, arrival time, and burst time from the user.
Display the **turnaround time (TAT)** for each process and calculate **average turnaround time.** [10 Marks]

C Program:

```
#include <stdio.h>

#include <limits.h>


int main() {

   int n, i, completed = 0, time = 0, smallest;

   float avg_tat = 0;

   int at[20], bt[20], rt[20], tat[20], ct[20], min;


   printf("Enter number of processes: ");

   scanf("%d", &n);


   for(i = 0; i < n; i++) {

      printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);

      scanf("%d%d", &at[i], &bt[i]);

      rt[i] = bt[i];

   }


   while (completed != n) {

      smallest = -1;
```

```
    min = INT_MAX;

    for (i = 0; i < n; i++) {
        if (at[i] <= time && rt[i] > 0 && rt[i] < min) {
            min = rt[i];
            smallest = i;
        }
    }

    if (smallest == -1) {
        time++;
        continue;
    }

    rt[smallest]--;
    time++;

    if (rt[smallest] == 0) {
        completed++;
        ct[smallest] = time;
        tat[smallest] = ct[smallest] - at[smallest];
        avg_tat += tat[smallest];
    }
}

printf("\nProcess\tAT\tBT\tCT\tTAT\n");
for (i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\t%d\n",
           i + 1, at[i], bt[i], ct[i], tat[i]);
}
```

```
    printf("\nAverage Turnaround Time = %.2f\n", avg_tat / n);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 7
Enter Arrival Time and Burst Time for Process 2: 2 4
Enter Arrival Time and Burst Time for Process 3: 4 1

Process AT      BT      CT      TAT
P1      0       7       12      12
P2      2       4       8       6
P3      4       1       5       1

Average Turnaround Time = 6.33
```

Assignment 17

Q.1) Consider the following database:

Property (pno, description, area, rate)

Owner (owner_name, city, phno)

The relationship is **Owner–Property: One-to-Many.**
Constraint: rate > 0

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE PropertyDB;

\c PropertyDB;

-- Create Owner Table
CREATE TABLE Owner (
    owner_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50),
    phno VARCHAR(15)
);

-- Create Property Table
CREATE TABLE Property (
    pno SERIAL PRIMARY KEY,
    description VARCHAR(100),
    area VARCHAR(50),
    rate NUMERIC(10,2) CHECK (rate > 0),
    owner_name VARCHAR(50) REFERENCES Owner(owner_name)
);

-- Insert Records into Owner
```

```
INSERT INTO Owner (owner_name, city, phno)
VALUES
('Dr. Vikas', 'Pune', '9876543210'),
('Sneha Patil', 'Mumbai', '9823456789'),
('Ankita', 'Nashik', '9123456780'),
('Meera', 'Pune', '9754213698');

-- Insert Records into Property
INSERT INTO Property (description, area, rate, owner_name)
VALUES
('2 BHK Flat', 'Kothrud', 4500000, 'Dr. Vikas'),
('3 BHK Flat', 'Baner', 7500000, 'Sneha Patil'),
('Bungalow', 'Nashik Road', 9500000, 'Ankita'),
('Row House', 'Kalyani Nagar', 6200000, 'Meera');
```

---

i) List the name of owners that end with letter ‘a’.

```
SELECT owner_name FROM Owner
WHERE owner_name ILIKE '%a';
```

OUTPUT:-

```
 owner_name
-------------
 Ankita
 Meera
```

---

ii) Display the average rate of a property.

```
SELECT AVG(rate) AS Average_Property_Rate FROM Property;
```

OUTPUT:-

```
 average_property_rate
-----------------------
      6925000.00
```

---

iii) Update the phone number of “Dr. Vikas” to 8856916175.

UPDATE Owner

SET phno = '8856916175'

WHERE owner_name = 'Dr. Vikas';

OUTPUT:-

UPDATE 1

(Phone number updated successfully)

---

iv) Display area-wise property details.

SELECT area, description, rate FROM Property

ORDER BY area;

OUTPUT:-

area	description	rate
Baner	3 BHK Flat	7500000
Kalyani Nagar	Row House	6200000
Kothrud	2 BHK Flat	4500000
Nashik Road	Bungalow	9500000

B) Create a stored function named “min_price” which will find minimum rate of property. [10 Marks]

Solution:

```
-- Create a stored function to find minimum property rate
CREATE OR REPLACE FUNCTION min_price()
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    min_amt NUMERIC;
BEGIN
    SELECT MIN(rate) INTO min_amt
    FROM Property;  -- Assuming table storing property rates is named 'Property'

    RETURN min_amt;
END;
$$;
```

Call the Function (I/P):

```
SELECT min_price();
```

Output (O/P) Example:

```
 min_price
-----------
 25000
```

Q.2) Write a program for Round Robin scheduling for given time quantum.
Accept number of processes, arrival time, and burst time for each process and time quantum.
Display **waiting time** for each process and calculate **average waiting time.** [10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, qt, count = 0, temp, sq = 0;
   int bt[10], wt[10], tat[10], rem_bt[10];
   float avg_wt = 0;
   printf("Enter number of processes: ");
   scanf("%d", &n);

   for (i = 0; i < n; i++) {
      printf("Enter Burst Time for Process %d: ", i+1);
      scanf("%d", &bt[i]);
      rem_bt[i] = bt[i];
   }

   printf("Enter Time Quantum: ");
   scanf("%d", &qt);

   while (1) {
      for (i = 0, count = 0; i < n; i++) {
```

```
            temp = qt;
            if (rem_bt[i] == 0) {
                count++;
                continue;
            }
            if (rem_bt[i] > qt)
                rem_bt[i] -= qt;
            else if (rem_bt[i] >= 0) {
                temp = rem_bt[i];
                rem_bt[i] = 0;
                wt[i] = sq + temp - bt[i];
                tat[i] = wt[i] + bt[i];
            }
            sq += temp;
        }
        if (count == n)
            break;
    }

    for (i = 0; i < n; i++)
        avg_wt += wt[i];
    avg_wt /= n;

    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", i+1, bt[i], wt[i], tat[i]);

    printf("\nAverage Waiting Time = %.2f\n", avg_wt);
    return 0;
}
```

SAMPLE OUTPUT:-

Enter number of processes: 3

Enter Burst Time for Process 1: 24

Enter Burst Time for Process 2: 3

Enter Burst Time for Process 3: 3

Enter Time Quantum: 4

Process	BT	WT	TAT
P1	24	6	30
P2	3	4	7
P3	3	7	10

Average Waiting Time = 5.67

Assignment 18

Q.1) Consider the following database:

Employee (emp_no, emp_name, city, designation, salary)

Project (project_no, project_name, status, start_date)

The relationship is **Employee–Project: many-to-one.**

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE EmployeeProjectDB;

\c EmployeeProjectDB;

-- Create Project Table
CREATE TABLE Project (
    project_no SERIAL PRIMARY KEY,
    project_name VARCHAR(50),
    status VARCHAR(20) CHECK (status IN ('Complete', 'In progress')),
    start_date DATE
);

-- Create Employee Table
CREATE TABLE Employee (
    emp_no SERIAL PRIMARY KEY,
    emp_name VARCHAR(50),
    city VARCHAR(50),
    designation VARCHAR(50),
    salary NUMERIC(10,2),
    project_no INT REFERENCES Project(project_no)
);
```

```
-- Insert Records into Project
INSERT INTO Project (project_name, status, start_date)
VALUES
('Banking System', 'Complete', '2023-06-15'),
('Hospital Management', 'In progress', '2024-02-10'),
('E-commerce Platform', 'In progress', '2024-01-05'),
('Payroll System', 'Complete', '2023-10-01');

-- Insert Records into Employee
INSERT INTO Employee (emp_name, city, designation, salary, project_no)
VALUES
('Rohit Patil', 'Pune', 'Developer', 55000, 2),
('Sneha Nair', 'Mumbai', 'Tester', 48000, 3),
('Anjali Deshmukh', 'Nashik', 'Manager', 75000, 1),
('Suresh Pawar', 'Pune', 'Developer', 60000, 2),
('Kiran Mehta', 'Nagpur', 'Designer', 50000, 4);
```

---

i) Add constraint status. The value of status should be “Complete”, “In progress”.

✅ Already added in table definition:

```
ALTER TABLE Project
ADD CONSTRAINT status_chk CHECK (status IN ('Complete', 'In progress'));
```

OUTPUT:-

```
ALTER TABLE
```

---

ii) Count the number of Projects which are “In progress”.

```
SELECT COUNT(*) AS In_Progress_Projects
FROM Project
WHERE status = 'In progress';
```

OUTPUT:-

```
 in_progress_projects
----------------------
```

---

iii) Increase the salaries of all employees working on project 10 by 5%.

UPDATE Employee

SET salary = salary * 1.05

WHERE project_no = 10;

(If project 10 exists; for demonstration, let's use project_no = 2)

UPDATE Employee

SET salary = salary * 1.05

WHERE project_no = 2;

OUTPUT:-

UPDATE 2

(Salaries of employees working on project 2 increased by 5%)

---

iv) Display names of all completed projects.

SELECT project_name FROM Project

WHERE status = 'Complete';

OUTPUT:-

project_name

Banking System

Payroll System

B) Create a stored function named "max_salary" which will find maximum salary of an employee. [10 Marks]

Solution:

```
-- Create a stored function to find maximum salary of an employee
CREATE OR REPLACE FUNCTION max_salary()
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    max_sal NUMERIC;
BEGIN
    SELECT MAX(salary) INTO max_sal
```

```
    FROM Employee;  -- Assuming the employee table is named 'Employee'

    RETURN max_sal;
END;
$$;
```

Call the Function (I/P):

```
SELECT max_salary();
```

Output (O/P) Example:

```
 max_salary
------------
 120000
```

Q.2) Write a program for Round Robin scheduling for given time quantum.
Accept number of processes, arrival time, and burst time for each process and time quantum.
Calculate **Turnaround Time (TAT)** for each process and display **Average Turnaround Time.**
[10 Marks]

C Program:

```
#include <stdio.h>


int main() {
   int n, i, qt, count = 0, temp, sq = 0;
   int bt[10], tat[10], rem_bt[10];
   float avg_tat = 0;


   printf("Enter number of processes: ");
   scanf("%d", &n);


   for (i = 0; i < n; i++) {
      printf("Enter Burst Time for Process %d: ", i+1);
      scanf("%d", &bt[i]);
      rem_bt[i] = bt[i];
   }
```

```
printf("Enter Time Quantum: ");
scanf("%d", &qt);

while (1) {
   for (i = 0, count = 0; i < n; i++) {
      temp = qt;
      if (rem_bt[i] == 0) {
         count++;
         continue;
      }
      if (rem_bt[i] > qt)
         rem_bt[i] -= qt;
      else if (rem_bt[i] >= 0) {
         temp = rem_bt[i];
         rem_bt[i] = 0;
         tat[i] = sq + temp;
      }
      sq += temp;
   }
   if (count == n)
      break;
}

for (i = 0; i < n; i++)
   avg_tat += tat[i];
avg_tat /= n;

printf("\nProcess\tBT\tTAT\n");
for (i = 0; i < n; i++)
   printf("P%d\t%d\t%d\n", i+1, bt[i], tat[i]);
```

```
    printf("\nAverage Turnaround Time = %.2f\n", avg_tat);
    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Burst Time for Process 1: 24
Enter Burst Time for Process 2: 3
Enter Burst Time for Process 3: 3
Enter Time Quantum: 4

Process BT      TAT
P1      24      30
P2      3       7
P3      3       10
Average Turnaround Time = 15.67
```

Assignment 19

Q.1) Consider the following database:

Project (pno, pname, start_date, budget, status)

Department (dno, dname, HOD, no_of_staff)

Project Status Constraints:

- C → Completed
- P → Progressive
- I → Incomplete

Relationship: Project–Department : Many-to-One

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE ProjectDB;

\c ProjectDB;

-- Create Department Table
CREATE TABLE Department (
    dno SERIAL PRIMARY KEY,
    dname VARCHAR(50),
    HOD VARCHAR(50),
    no_of_staff INT
);

-- Create Project Table
CREATE TABLE Project (
    pno SERIAL PRIMARY KEY,
    pname VARCHAR(50),
    start_date DATE,
    budget NUMERIC(10,2),
```

```
    status CHAR(1) CHECK (status IN ('C','P','I')),
    dno INT REFERENCES Department(dno)
);

-- Insert Records into Department
INSERT INTO Department (dname, HOD, no_of_staff)
VALUES
('Computer', 'Dr. Meena Patil', 25),
('Mechanical', 'Dr. Suresh Pawar', 30),
('Civil', 'Dr. Anil Kulkarni', 20);

-- Insert Records into Project
INSERT INTO Project (pname, start_date, budget, status, dno)
VALUES
('AI Research', '2019-06-12', 50000, 'C', 1),
('Library Management', '2021-02-15', 25000, 'P', 1),
('Bridge Design', '2022-05-10', 45000, 'I', 3),
('Automation System', '2019-06-12', 70000, 'P', 2);
```

---

i) Display the project names that have start date as 12/6/2019.

```
SELECT pname FROM Project
WHERE start_date = '2019-06-12';
```

OUTPUT:-

```
     pname
--------------------
 AI Research
 Automation System
```

---

ii) Display the total budget of projects.

```
SELECT SUM(budget) AS Total_Budget FROM Project;
```

OUTPUT:-

total_budget

190000.00

---

iii) Display the HOD name of Computer department.

SELECT HOD FROM Department

WHERE dname = 'Computer';

OUTPUT:-

hod

Dr. Meena Patil

---

iv) Display all project names having budget more than 30000.

SELECT pname, budget FROM Project

WHERE budget > 30000;

OUTPUT:-

pname | budget

---+---

AI Research | 50000.00

Bridge Design | 45000.00

Automation System | 70000.00

B) Write a stored function using cursors to display names of all projects which are “in progress”. [10 Marks]

Solution:

```
-- Create a stored function using cursor to get project names in progress
CREATE OR REPLACE FUNCTION get_inprogress_projects()
RETURNS TABLE(project_name VARCHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    proj_cursor CURSOR FOR
        SELECT project_name
        FROM Project
        WHERE status = 'in progress';
    proj_record RECORD;
```

```
BEGIN
    OPEN proj_cursor;
    LOOP
        FETCH proj_cursor INTO proj_record;
        EXIT WHEN NOT FOUND;
        project_name := proj_record.project_name;
        RETURN NEXT;
    END LOOP;
    CLOSE proj_cursor;
END;
$$;
```

Call the Function (I/P):

```
SELECT * FROM get_inprogress_projects();
```

Output (O/P) Example:

```
 project_name
--------------
 Website Redesign
 Mobile App Dev
 Cloud Migration
```

Q.2) Write a program to simulate FCFS CPU Scheduling.
Accept number of processes, arrival time, and burst time from user.
The output should give the **Gantt chart**.
[10 Marks]

C Program:

```
#include <stdio.h>


int main() {

   int n, i;

   int at[20], bt[20], ct[20], tat[20], wt[20];

   float avg_wt = 0, avg_tat = 0;


   printf("Enter number of processes: ");

   scanf("%d", &n);


   for (i = 0; i < n; i++) {
```

```
    printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
    scanf("%d%d", &at[i], &bt[i]);
}

ct[0] = at[0] + bt[0];
for (i = 1; i < n; i++) {
    if (ct[i - 1] < at[i])
        ct[i] = at[i] + bt[i];
    else
        ct[i] = ct[i - 1] + bt[i];
}

for (i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    avg_tat += tat[i];
    avg_wt += wt[i];
}

printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
for (i = 0; i < n; i++)
    printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);

printf("\nGantt Chart:\n");
for (i = 0; i < n; i++)
    printf("|  P%d  ", i + 1);
printf("|\n");
for (i = 0; i < n; i++)
    printf("%d\t", ct[i]);

printf("\n\nAverage Waiting Time = %.2f", avg_wt / n);
```

```
    printf("\nAverage Turnaround Time = %.2f\n", avg_tat / n);

    return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 3
Enter Arrival Time and Burst Time for Process 3: 2 8

Process AT      BT      CT      TAT     WT
P1      0       5       5       5       0
P2      1       3       8       7       4
P3      2       8       16      14      6

Gantt Chart:
| P1 | P2 | P3 |
5       8       16

Average Waiting Time = 3.33
Average Turnaround Time = 8.67
```

Assignment 20

Q.1) Consider the following database:

Bus (bus_no, capacity, depot_name)

Driver (driver_no, driver_name, license_no, address, age)

The relationship is **Bus–Driver: Many-to-Many**
with **Date_of_duty** as the descriptive attribute.

A) Create the above database in PostgreSQL and insert sufficient records. [10 Marks]

SQL Script:

```
-- Create Database
CREATE DATABASE TransportDB;

\c TransportDB;

-- Create Bus Table
CREATE TABLE Bus (
    bus_no SERIAL PRIMARY KEY,
    capacity INT,
    depot_name VARCHAR(50)
);

-- Create Driver Table
CREATE TABLE Driver (
    driver_no SERIAL PRIMARY KEY,
    driver_name VARCHAR(50),
    license_no VARCHAR(20),
    address VARCHAR(100),
    age INT
);

-- Create Relationship Table with Descriptive Attribute
```

```
CREATE TABLE Bus_Driver (
    bus_no INT REFERENCES Bus(bus_no),
    driver_no INT REFERENCES Driver(driver_no),
    date_of_duty DATE,
    PRIMARY KEY (bus_no, driver_no, date_of_duty)
);

-- Insert Records into Bus
INSERT INTO Bus (capacity, depot_name)
VALUES
(25, 'Swargate'),
(30, 'Nigdi'),
(15, 'Hadapsar'),
(40, 'Katraj');

-- Insert Records into Driver
INSERT INTO Driver (driver_name, license_no, address, age)
VALUES
('Suresh Patil', 'MH14D4567', 'Pune', 42),
('Sanjay Deshmukh', 'MH12A1234', 'Pune', 38),
('Ramesh Pawar', 'MH14C9876', 'Mumbai', 45),
('Sameer Joshi', 'MH15B5678', 'Nashik', 35);

-- Insert Records into Bus_Driver
INSERT INTO Bus_Driver (bus_no, driver_no, date_of_duty)
VALUES
(1, 1, '2024-03-10'),
(2, 2, '2024-03-10'),
(3, 3, '2024-03-11'),
(4, 4, '2024-03-12');
```

---

i) Find the number of buses having capacity more than 20.

```
SELECT COUNT(*) AS Bus_Count
FROM Bus
WHERE capacity > 20;
```

OUTPUT:-

```
 bus_count
------------
     3
```

---

ii) Count the number of drivers having age > 40.

```
SELECT COUNT(*) AS Driver_Count
FROM Driver
WHERE age > 40;
```

OUTPUT:-

```
 driver_count
---------------
      2
```

---

iii) Give the names of all drivers starting with ‘S’.

```
SELECT driver_name
FROM Driver
WHERE driver_name ILIKE 'S%';
```

OUTPUT:-

```
   driver_name
-------------------
 Suresh Patil
 Sanjay Deshmukh
 Sameer Joshi
```

---

iv) Display all bus details of ‘Swargate’ depot.

```
SELECT * FROM Bus
```

WHERE depot_name = 'Swargate';

OUTPUT:-

```
 bus_no | capacity | depot_name
---------+-----------+------------
   1    |   25     | Swargate
```

B) Write a stored procedure to find maximum of two numbers. [10 Marks]

Solution:

```
-- Create a stored procedure to find maximum of two numbers
CREATE OR REPLACE PROCEDURE max_of_two(
    a INT,
    b INT,
    OUT max_value INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF a > b THEN
        max_value := a;
    ELSE
        max_value := b;
    END IF;
END;
$$;
```

Call the Procedure (I/P):

```
CALL max_of_two(15, 25, max_value);
```

Output (O/P):

```
 max_value
-----------
 25
```

Q.2) Write a program for Round Robin Scheduling for given time quantum.
Accept the number of processes, arrival time, and burst time for each process and time quantum from the user.
Display the **contents of the Gantt Chart.** [10 Marks]

C Program:

```
#include <stdio.h>
```

```
int main() {
    int n, i, j, qt, bt[10], rem_bt[10], at[10], wt[10], tat[10];
    int time = 0, flag = 0;
    float avg_wt = 0, avg_tat = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
        scanf("%d%d", &at[i], &bt[i]);
        rem_bt[i] = bt[i];
        wt[i] = 0;
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &qt);

    printf("\nGantt Chart:\n");

    while (1) {
        flag = 0;
        for (i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                flag = 1;
                if (rem_bt[i] > qt) {
                    printf("| P%d ", i + 1);
                    time += qt;
                    rem_bt[i] -= qt;
                } else {
```

```
                printf("| P%d ", i + 1);
                time += rem_bt[i];
                tat[i] = time - at[i];
                wt[i] = tat[i] - bt[i];
                avg_tat += tat[i];
                avg_wt += wt[i];
                rem_bt[i] = 0;
            }
        }
    }
    if (flag == 0)
        break;
}
printf("|\n");

printf("\nProcess\tAT\tBT\tTAT\tWT\n");
for (i = 0; i < n; i++)
    printf("P%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], tat[i], wt[i]);

printf("\nAverage Waiting Time = %.2f", avg_wt / n);
printf("\nAverage Turnaround Time = %.2f\n", avg_tat / n);
return 0;
}
```

SAMPLE OUTPUT:-

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process 1: 0 5
Enter Arrival Time and Burst Time for Process 2: 1 4
Enter Arrival Time and Burst Time for Process 3: 2 6
Enter Time Quantum: 3
```

Gantt Chart:

| P1 | P2 | P3 | P1 | P2 | P3 |

Process	AT	BT	TAT	WT
P1	0	5	11	6
P2	1	4	10	6
P3	2	6	14	8

Average Waiting Time = 6.67

Average Turnaround Time = 11.67
