SLIP 21 - CREATE TABLE Customer (

cust_no INT PRIMARY KEY,

cust_name VARCHAR(50),

city VARCHAR(50)

);

CREATE TABLE Loan (

loan_no INT PRIMARY KEY,

loan_amt NUMERIC CHECK (loan_amt > 0)

);

-- Many-to-Many relation

CREATE TABLE Customer_Loan (

cust_no INT REFERENCES Customer(cust_no),

loan_no INT REFERENCES Loan(loan_no),

PRIMARY KEY (cust_no, loan_no)

);

INSERT INTO Customer VALUES

(1, 'Amit Shah', 'Pune'),

(2, 'Anita Joshi', 'Mumbai'),

(3, 'Rahul Patil', 'Delhi'),

(4, 'Arun Kale', 'Pune');

INSERT INTO Loan VALUES

(101, 150000),

(102, 350000),

(103, 500000),

(104, 250000);

INSERT INTO Customer_Loan VALUES

(1, 101),

(1, 103),

(2, 102),

(3, 104),

(4, 101);

) List all customers whose name starts with 'A'

```
SELECT * FROM Customer
WHERE cust_name LIKE 'A%';
```

ii) Display city-wise customer names

```
SELECT city, cust_name
FROM Customer
ORDER BY city, cust_name;
```

iii) Display all loan numbers where amount > 2 lakhs

```
SELECT loan_no, loan_amt
FROM Loan
WHERE loan_amt > 200000;
```

iv) Change city 'Pune' → 'Mumbai' for a specific customer

(Example: Amit Shah)

```
UPDATE Customer
SET city = 'Mumbai'
WHERE cust_name = 'Amit Shah' AND city = 'Pune';
```

✅ **B) Stored Function Using Cursor – Sorted by City**

Stored Function

```
CREATE OR REPLACE FUNCTION display_customers_cursor()
RETURNS void AS $$
DECLARE
   cur CURSOR FOR
      SELECT cust_no, cust_name, city
      FROM Customer
      ORDER BY city;
```

```
    r RECORD;
BEGIN
    OPEN cur;

    LOOP
        FETCH cur INTO r;
        EXIT WHEN NOT FOUND;

        RAISE NOTICE 'Cust No: %, Name: %, City: %',
                r.cust_no, r.cust_name, r.city;
    END LOOP;

    CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

Execution

```
SELECT display_customers_cursor();
```

SLIP – 22

1. Create Tables

```
CREATE TABLE Customer (
   cust_no INT PRIMARY KEY,
   cust_name VARCHAR(50),
   city VARCHAR(50)
);

CREATE TABLE Product (
   product_no INT PRIMARY KEY,
   pname VARCHAR(50),
   price NUMERIC CHECK (price > 0)
);

-- Many-to-Many Relationship
CREATE TABLE Customer_Product (
   cust_no INT REFERENCES Customer(cust_no),
   product_no INT REFERENCES Product(product_no),
   PRIMARY KEY (cust_no, product_no)
);
```

---

2. Insert Sample Records

```
INSERT INTO Customer VALUES
(1, 'Rekha', 'Pune'),
(2, 'Megha', 'Mumbai'),
(3, 'Aruna', 'Delhi'),
(4, 'Sonia', 'Pune');

INSERT INTO Product VALUES
(101, 'Laptop', 50000),
(102, 'Keyboard', 900),
```

(103, 'Mobile', 15000),

(104, 'Mouse', 1200);

INSERT INTO Customer_Product VALUES

(1, 101),

(1, 103),

(2, 104),

(3, 101),

(4, 103);

✅ **Execute Queries**

i) List all customers whose name ends with 'A'

SELECT * FROM Customer

WHERE cust_name LIKE '%A';

---

ii) Count number of products whose price > 1000

SELECT COUNT(*) AS expensive_products

FROM Product

WHERE price > 1000;

---

iii) Increase price of all products by 5%

UPDATE Product

SET price = price * 1.05;

---

iv) Display details of customers who are from a given city

(Example: Pune)

SELECT * FROM Customer

WHERE city = 'Pune';

✅ **B) Stored Procedure “addrecords” to Insert Customer**

Stored Procedure

```
CREATE OR REPLACE PROCEDURE addrecords(
    p_cust_no INT,
    p_cust_name VARCHAR,
    p_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Customer(cust_no, cust_name, city)
    VALUES (p_cust_no, p_cust_name, p_city);

    RAISE NOTICE 'Customer added successfully.';
END;
$$;
```

Execute Procedure

```
CALL addrecords(5, 'Ria', 'Nagpur');
```

✅ **Q.2) OS Program – Non-Preemptive SJF (Shortest Job First)**

Program that accepts processes, arrival time, burst time, and displays Gantt Chart.

C Program

```
#include <stdio.h>

int main() {
    int n, i, j, t = 0, completed = 0, min_index;
    float total_wait = 0, total_tat = 0;

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

    int at[n], bt[n], pid[n], wt[n], tat[n], done[n];
```

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

printf("\nGANTT CHART\n");
printf("0");

while (completed != n) {
    min_index = -1;

    for (i = 0; i < n; i++) {
        if (!done[i] && at[i] <= t) {
            if (min_index == -1 || bt[i] < bt[min_index])
                min_index = i;
        }
    }

    if (min_index == -1) {
        t++;
        continue;
    }

    t += bt[min_index];
    wt[min_index] = t - at[min_index] - bt[min_index];
    tat[min_index] = t - at[min_index];
    done[min_index] = 1;
    completed++;
```

```
        printf(" | P%d | %d", pid[min_index], t);
    }

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

    return 0;
}
```

SLIP – 23

A) Create Database, Tables & Insert Records

1. Create Tables

```
CREATE TABLE Student (
   rno INT PRIMARY KEY,
   name VARCHAR(50),
   city VARCHAR(50)
);

CREATE TABLE Subject (
   subno INT PRIMARY KEY,
   subname VARCHAR(50),
   teachername VARCHAR(50)
);

-- Many-to-Many with descriptive attribute "mark"
CREATE TABLE Student_Subject (
   rno INT REFERENCES Student(rno),
   subno INT REFERENCES Subject(subno),
   mark INT,
   PRIMARY KEY (rno, subno)
);
```

---

2. Insert Sample Records

```
INSERT INTO Student VALUES
(1, 'Rohit', 'Pune'),
(2, 'Meena', 'Mumbai'),
(3, 'Asha', 'Pune'),
(4, 'Sanjay', 'Delhi');

INSERT INTO Subject VALUES
```

(101, 'OS', 'Dr. Kulkarni'),

(102, 'DBMS', 'Prof. Gupta'),

(103, 'Networking', 'Dr. Kulkarni');

INSERT INTO Student_Subject VALUES

(1, 101, 78),

(1, 102, 80),

(2, 103, 69),

(3, 101, 88),

(4, 102, 75);

✅ **Execute Queries**

i) List all students from a given city

(Example: Pune)

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

---

ii) Count number of subjects taught by a particular teacher

(Example: Dr. Kulkarni)

```
SELECT COUNT(*) AS subject_count
FROM Subject
WHERE teachername = 'Dr. Kulkarni';
```

---

iii) Display name of all teachers who teach “OS”

```
SELECT teachername
FROM Subject
WHERE subname = 'OS';
```

---

iv) Delete record of a student by name

(Example: Asha)

```
DELETE FROM Student
```

```
WHERE name = 'Asha';
```

✅ **B) Stored Procedure “addrecords” to Add Student**

Stored Procedure

```
CREATE OR REPLACE PROCEDURE addrecords(
    p_rno INT,
    p_name VARCHAR,
    p_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Student(rno, name, city)
    VALUES (p_rno, p_name, p_city);

    RAISE NOTICE 'Student record added successfully.';
END;
$$;
```

Execute Procedure

```
CALL addrecords(5, 'Vikas', 'Nagpur');
```

✅ **Q.2 – Round Robin Scheduling Program (Gantt Chart)**

Language: C (simple and exam-ready)

```
#include <stdio.h>

int main() {
    int n, tq, i, t = 0, done;

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

```
int at[n], bt[n], rt[n], pid[n];

for (i = 0; i < n; i++) {
   pid[i] = i + 1;
   printf("Enter arrival time and burst time of P%d: ", i + 1);
   scanf("%d %d", &at[i], &bt[i]);
   rt[i] = bt[i];
}

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

printf("\nGANTT CHART\n");
printf("0");

do {
   done = 1;

   for (i = 0; i < n; i++) {
      if (rt[i] > 0 && at[i] <= t) {
         done = 0;

         if (rt[i] > tq) {
            rt[i] -= tq;
            t += tq;
            printf(" | P%d | %d", pid[i], t);
         } else {
            t += rt[i];
            rt[i] = 0;
            printf(" | P%d | %d", pid[i], t);
         }
```

```
            }
        }
    } while (!done);

    return 0;
}
```

SLIP – 24

1. Create Tables

```
CREATE TABLE Author (
   aid INT PRIMARY KEY,
   aname VARCHAR(50),
   mobile VARCHAR(15),
   city VARCHAR(50)
);

CREATE TABLE Book (
   bid INT PRIMARY KEY,
   btitle VARCHAR(100),
   price NUMERIC,
   publication VARCHAR(100),
   aid INT REFERENCES Author(aid)   -- One author → many books
);
```

---

2. Insert Sample Records

```
INSERT INTO Author VALUES
(1, 'Sanjay Patil', '9876543210', 'Pune'),
(2, 'Sumit Rao', '9988776655', 'Mumbai'),
(3, 'Anita Shah', '9123456789', 'Delhi');

INSERT INTO Book VALUES
(101, 'Operating Systems', 550, 'Prentice Hall', 1),
(102, 'DBMS Concepts', 650, 'McGraw Hill', 2),
(103, 'Computer Networks', 700, 'Prentice Hall', 1),
(104, 'C Programming', 400, 'TechWorld', 3);
```

✅ **Execute Queries**

i) Display author names starting with ‘S’

```
SELECT aname
FROM Author
WHERE aname LIKE 'S%';
```

---

ii) Total price of books published by “Prentice Hall”

```
SELECT SUM(price) AS total_price
FROM Book
WHERE publication = 'Prentice Hall';
```

---

iii) Update mobile number of author (example: Sumit Rao)

```
UPDATE Author
SET mobile = '9844567822'
WHERE aname = 'Sumit Rao';
```

---

iv) Display details of books written by a given author

(Example: Sanjay Patil)

```
SELECT b.*
FROM Book b
JOIN Author a ON b.aid = a.aid
WHERE a.aname = 'Sanjay Patil';
```

✅ **B) Stored Function “max_price” (returns maximum book price)**

Function

```
CREATE OR REPLACE FUNCTION max_price()
RETURNS NUMERIC
AS $$
DECLARE
   maxp NUMERIC;
BEGIN
   SELECT MAX(price) INTO maxp FROM Book;
   RETURN maxp;
```

```
END;
$$ LANGUAGE plpgsql;
```

Execute

```
SELECT max_price();
```

✅ **Q.2 – Operating System Program: Banker's Algorithm**

C Program – Exam-Ready Implementation

```
#include <stdio.h>

int main() {
    int p, r, i, j;

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

    printf("Enter number of resources: ");
    scanf("%d", &r);

    int max[p][r], alloc[p][r], need[p][r], avail[r];

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

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

    printf("\nEnter Available Resources:\n");
```

```
    for (j = 0; j < r; j++)
        scanf("%d", &avail[j]);

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

    printf("\nBanker's Algorithm Completed.\n");
    return 0;
}
```

SLIP – 25

1. Create Tables

```
CREATE TABLE Department (
   dno INT PRIMARY KEY,
   dname VARCHAR(50),
   location VARCHAR(50)
);

CREATE TABLE Professor (
   prof_no INT PRIMARY KEY,
   prof_name VARCHAR(50),
   designation VARCHAR(50),
   salary NUMERIC,
   dno INT REFERENCES Department(dno)
);
```

---

2. Insert Sample Records

```
INSERT INTO Department VALUES
(10, 'Computer', 'Pune'),
(20, 'Mechanical', 'Mumbai'),
(30, 'Electrical', 'Pune');

INSERT INTO Professor VALUES
(1, 'Rahul Kumar', 'Associate Prof', 75000, 10),
(2, 'Sameer', 'Professor', 90000, 10),
(3, 'Anwar', 'Assistant Prof', 65000, 20),
(4, 'Sohail', 'Professor', 100000, 30),
(5, 'Vijay Kumar', 'Lecturer', 55000, 10);
```

✅ **Execute Queries**

i) Display average salary of professor

```
SELECT AVG(salary) AS average_salary
FROM Professor;
```

---

ii) List details of all departments located at a given location

(Example: Pune)

```
SELECT *
FROM Department
WHERE location = 'Pune';
```

---

iii) Display details of professors whose names end with 'r'

```
SELECT *
FROM Professor
WHERE prof_name LIKE '%r';
```

---

iv) Display details of all professors working in “Computer” department

```
SELECT p.*
FROM Professor p
JOIN Department d ON p.dno = d.dno
WHERE d.dname = 'Computer';
```

✅ **B) Stored Procedure: display_message**

Stored Procedure

```
CREATE OR REPLACE PROCEDURE display_message()
LANGUAGE plpgsql
AS $$
BEGIN
   RAISE NOTICE 'Welcome to RDBMS world!!!!';
END;
$$;
```

Execute

```
CALL display_message();
```

✅ **Q.2 – Program to Display Allocation, Max, Need (Banker's Algorithm Part)**

(Using given data)

✓ **We calculate Need = Max – Allocation**

C Program (Simple Journal-Ready Version)

```
#include <stdio.h>

int main() {
   int i, j;

   int alloc[5][3] = {
      {0, 1, 0},
      {2, 0, 0},
      {3, 0, 2},
      {2, 1, 1},
      {0, 0, 2}
   };

   int max[5][3] = {
      {7, 5, 3},
      {3, 2, 2},
      {9, 0, 2},
      {2, 2, 2},
      {4, 3, 3}
   };

   int avail[3] = {3, 3, 2};

   int need[5][3];

   // Calculate Need matrix
```

```
    for (i = 0; i < 5; i++)
        for (j = 0; j < 3; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    printf("Allocation Matrix:\n");
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 3; j++)
            printf("%d ", alloc[i][j]);
        printf("\n");
    }

    printf("\nMax Matrix:\n");
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 3; j++)
            printf("%d ", max[i][j]);
        printf("\n");
    }

    printf("\nAvailable Resources:\n");
    for (j = 0; j < 3; j++)
        printf("%d ", avail[j]);

    printf("\n\nNeed Matrix:\n");
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 3; j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }

    return 0;
}
```

Q.2) OS Program — Pre-emptive Shortest Job First (SJF) Scheduling

C Program with Gantt Chart Output

```
#include <stdio.h>

int main() {
   int n, completed = 0, t = 0, i, min;
   float total_wait = 0, total_tat = 0;

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

   int at[n], bt[n], rt[n], wt[n], tat[n], pid[n];

   for (i = 0; i < n; i++) {
      pid[i] = i + 1;
      printf("Enter arrival time and burst time for P%d: ", i + 1);
      scanf("%d %d", &at[i], &bt[i]);
      rt[i] = bt[i];
   }

   printf("\nGANTT CHART\n");
   printf("0");

   while (completed != n) {
      min = -1;
      for (i = 0; i < n; i++) {
         if (at[i] <= t && rt[i] > 0) {
            if (min == -1 || rt[i] < rt[min])
               min = i;
         }
```

```
        }

        if (min == -1) {
            t++;
            continue;
        }

        rt[min]--;
        printf(" | P%d | %d", pid[min], t + 1);
        t++;

        if (rt[min] == 0) {
            completed++;
            tat[min] = t - at[min];
            wt[min] = tat[min] - bt[min];
            total_wait += wt[min];
            total_tat += tat[min];
        }
    }

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

    return 0;
}
```